BEHAVIOR-BASED ROBOT NAVIGATION IN UNKNOWN ENVIRONMENTS

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

by

RİFAT DÖVER

**IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING**

NICOSIA 2010

BEHAVIOR-BASED ROBOT NAVIGATION IN UNKNOWN ENVIRONMENTS

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

by

RİFAT DÖVER

In Partial Fulfillment of the Requirements for
the Degree of Master of Science
in
Computer Engineerıng

NICOSIA 2010

Rifat Döver : BEHAVIOR-BASED ROBOT NAVIGATION IN UNKNOWN ENVIRONMENTS

Approval of Director of Graduate School of Applied Sciences

Prof. Dr. İlkay SALİHOĞLU

We certify this thesis is satisfactory for the award of the degree of Master of Science in Computer Engineering

Examining Committee in Charge:

ABSTRACT

The problem of mobile robot navigation is a wide and complex problem. The environment which a robot moves in it may vary from static areas with fixed obstacles to fast changing dynamic areas with many moving dynamic obstacles. In these conditions navigation of mobile robot becomes important and most challenging problem. The complexity of mobile robot system can make the system cost intensive and high risk. Many researchers come with the solution of navigation problem by using the various types of control methods classified as global and local navigation methods. The most of software developed for these algorithms are working with fixed (statistic) obstacles, in two dimensional environments. In most cases the environment is characterized with fast-changing dynamic areas with many moving obstacles. For these reasons the development of navigation algorithms avoiding from static and at the same time from dynamical obstacles is becoming is very important problem. This study is an inquiry the question to real time obstacle avoidance and navigation problem that work in three-dimensional workspaces. Obstacle avoidance methods are based on a real-time control process. Sensors collect information of the environment that is processed by the navigation system of robot. Three dimensional mobile robot navigation approach equipped with rule-based navigation algorithm which uses behavior based approach to navigation and obstacle avoidance system is proposed. Mobile robot should navigated in dynamically changing environment and avoid from obstacle. Sensor sets provide necessary distance information to control system of a mobile robot. Using this information the proposed algorithm makes decision and navigate robot to the goal. Using Visual C++ (2008) the navigation algorithm proposed in 3D workspace developed.

Key words: dynamic, obstacle avoidance, navigation, 3 dimensional, simulation, software.

ÖZET

Mobil robotlarda yön bulma geniş ve karmaşık bir problemdir. Robotların hareket ettikleri çevreler değişmeyen statik alanlardaki sabit engeller ile hızla değişen dinamik alanlardaki dinamik engeller arasında değişkenlik göstermektedir. Bu durumda yön bulma mobil robotlar için önemli ve zor bir problem haline gelmektedir. Mobil robot sistemlerinin karmaşıklığı sistemi yüksek maliyetli ve riskli kılmaktadır. Birçok araştırmacı global ve lokal kontrol metotları kullanarak çözümler üretmişlerdir. Bu algoritmaları kullanarak oluşturulan yazılımların birçoğu sabit (statik) engeller kullanılarak iki boyutlu bir çevrede oluşturulmuştur. Birçok durumda çevre hızlı değişim gösteren dinamik alanlardaki hareketli objelerle karakterize edilir. Bu sebeple geliştirilecek yön bulma algoritması statik engellerle dinamik engellerden kaçmayı sağlayacak şekilde oluşturması çok büyük bir önem kazanmaktadır. Bu çalışma 3 boyutlu bir ortamda gerçek zamanlı engelden kaçma ve yön bulma sorunu hakkında bir çalışmadır. Engelden kaçma metotları gerçek zamanlı kontrol işlemleri ile yapılmıştır. Robot sensorlar yardımı ile çevre hakkında bilgi toplamakta ve bu yön bulma sistemi ile işlenmektedir. Davranışsal yaklaşım kullanan kural-tabanlı bir üç boyutlu robot yön bulma yaklaşımı ortaya konulmuştur. Robot dinamik olarak değişebilen durumlara ayak uydurabilecek tarzda yön bulmalı ve engellerden kaçınmalıdır. Sensorlar robotun kontrol sistemi için yeterli uzaklık bilgisini alabilmektedir. Bu bilgiyi kullanarak sunulan algoritma karar verip ve robotu hedefe doğru yönlendirmektedir. Üç boyutlu çalışma alanı ve yön bulma algoritması Visual C++ (2008) kullanılarak geliştirilmiştir.

Anahtar sözcükler: dinamik, engelden kaçma, yön bulma, 3 boyutlu, simulasyon.

ACKNOWLEDGEMENTS

First of all I am indebted to a number of individuals, who contributed directly or indirectly to the thesis text. In particular I would like to express my thanks to my supervisor Prof. Dr. Rahib Abiev for his help, valuable guideness, and patience.

Last but not the least, I am very grateful to the Near East University, its staff, students, and every one I met there.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Overview

In this chapter a robot navigation problem and its importance are shown. The navigation problems and its solution using Global and Local Navigation Methods are described. As an example the description of Local Navigation Methods such as Artificial Potential Field, Vector Field Histogram, Rule Based and Machine Learning Methods are presented. The steps of solution of Robot Navigation in this thesis are described.

1.2 Robot Navigation Problems

A robot is a versatile mechanical device –for example, a manipulator arm, a multijoint multi-fingered hand, a wheeled or legged vehicle, a free-flying platform, or a combination of these-equipped with actuators and sensors under the control of a computing system [1]. It operates in a workspace within the real world. This workspace is populated by physical objects and is subject to laws of nature. The robot performs tasks by executing motions in workspace [2].

Robotics has achieved its greatest success to date in the world of industrial manufacturing. Robot arms, or manipulators, comprise a 2 billion dollar industry [3]. Bolted at its shoulder to a specific position in the assembly line, the robot arm can move with great speed and accuracy to perform repetitive tasks such as spot welding and painting. In the electronics industry, manipulators place surface-mounted components with superhuman precision, making the portable telephone and laptop computer possible [3].

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.

Mobile robots come in two categories: holonomic and non-holonomic [5]. Holonomic robots are able to vary each component of their position and orientation independently. They are able to turn on the spot, and move in any direction regardless of orientation. Non-holonomic robots are not free to move in such manner. They may only be able to move in B limited number of directions depending on their orientation and/or they may not be able to change orientation without changing position. The vast majorities of robots are non-holonomic, and have different constraints on their motion. These constraints, together with the physical constraints like size and maximum speed, are known as the robot's kinematic constraints. Therefore it is important that a study should be conducted on the available mobile robot navigation techniques in order to clearly identify the potential contribution.

1.3 Local and Global Navigation Methodologies

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, such as infrared sensor, ultrasonic sensor, laser range finder, touch sensor, camera and so on can be used to detect the presence of obstacles.

Many efforts have been paid in the past to develop various obstacle avoidance algorithms. In general, the research in this field can be classified into two major areas: the *global navigation* [1, 2] and the *local navigation* [3, 5]. 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 robot navigation which are classified into local and Global. 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.

The most of upper mentioned approaches and software developed for these algorithms are working with fixed (statistic) obstacles, in two dimensional environments. In most cases the environment is characterized with fast-changing dynamic areas with many moving obstacles. The aim of this thesis is the development of robot navigation systems avoiding from static and at the same time from dynamical obstacles. Three dimensional a mobile robot navigation approach equipped with rule-based navigation algorithm which uses goal oriented approach to navigation and obstacle avoidance system is proposed.

Thesis consists of four chapters, conclusion, references and appendix.

Chapter two includes robot navigation problems and its solution using Global and Local Navigation Methods. The description of more used Local and Global Navigation Methods are described. Artificial Potential Field, Vector Field Histogram, Rule Based and Machine Learning Methods are presented. The review on Robot navigation is given.

Chapter three describes control architecture for development of autonomous mobile robot system. The behavior-based control using rule based approach is used to efficiently navigate the robot in order to avoid from conflicts and achieve a good performance.

Chapter four describes software realization of robot navigation. To build a real time dynamic obstacle avoidance simulation the real world with its obstacles is modeled. The different libraries are used for development of software. 3D Game Programming physics engines are used to simulate the real world effects. For this aim NVIDIA PhysX Engine SDK is used. Then using rule based approach the navigation algorithm is implemented for achieving goal.

Conclusion includes important results obtained from the thesis.

Appendix includes fragments of robot navigation programs.

2. REWIEW ON MOBILE ROBOT NAVIGATION

2.1 Overview

In this chapter robot navigation problems and its solution using Global and Local Navigation Methods are described. As an example the description of Local Navigation Methods such as Artificial Potential Field, Vector Field Histogram, Rule Based and Machine Learning Methods are presented. The review on Robot Navigation is given.

Most mobile robot navigation techniques are developed at two levels: Local level and Global level. Local navigation by reactive means is employed by techniques such as fuzzy systems, or field-based systems. In this a robot relies only on current or recent sensor data. This is useful for rapid responses to avoid collisions. However, this method of navigation tends to fail in the presence of convex or mobile obstacles. The robot will either repeatedly enter or leave the interior of the object (e.g. corridors), or encounter deadlocking constraints. Global navigation is used to break such deadlocks. Global navigation looks at the broader objective of the robot, and identifies long-range paths for the robot to follow. Global Navigation Techniques mostly static environment based techniques which are not so applicapable for mobile robot navigation. In this thesis concern as being more applicapable and dynamic environment based Local Navigation techniques takes the first place.

2.2 Robot Navigation Methods and Obstacle Collision 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 [7]. Others are Vector Field Histogram (VFH) 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: Rather than trying 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. They extended 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 modification to the techniques of mobile robot navigation 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: As an alternative to field-based methods, rule-based methods can be employed. 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 – Goal Seeking

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 [4].

Search Trees: Common techniques for building search trees come from Computer Science. These 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: However, 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 modeling 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: All the above mentioned techniques 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 [4].

Command Arbitration: The previously 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] “Distributed Architecture for Mobile Navigation”. In DAMN the robot is given various commands it can perform, like “move forward” and “turn right”. It is programmed with various behaviors, such as “avoid obstacle” and find goal”. Each behavior computes the best action for its current position and world model. Each behavior enters a vote for any actions it is willing to support with values indicating how much it supports each action.

A generalized form of this voting system is used in Howard's "Dynamic Utility Fusion" approach. Rather than sum votes for a single action to be executed immediately, the arbiter chooses longer strings of actions. The system then uses the behaviors to determine the utility (or vote) for each string [21]. The system then acts on the string actions with the highest votes within the search space.

Summation methods attempt to combine all the commands into a final executable command. In the Motor Schema system proposed by Arkin [22] the plans are produced as vectors, and the arbiter sums up the vectors to produce the executed action. This method can lead to a number of problems. The problems can be exhibited in situations where the force vectors cancel each other producing undesirable robot's movement. This makes summation methods less desirable than selection methods.

Hybrids-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.

Xiso [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.

2.3 Review on Local Navigation Methods

In this thesis the statement of software design of local navigation algorithm is considered. The most common used techniques in Local Navigation are Artificial Potential Fields Method, Vector Field Histograms, Local Adaptive Navigation and

Fuzzy Logic based Navigation Techniques. Hybrid of techniques gives the best results as it is mentioned below.

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 [25].

- 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.

Enxiu SHI and Junjie GUO [26] developed Analyzing of the disadvantage of Artificial Potential Field (APF) for mobile robot obstacle avoidance, local trap and vibrating, a new method for improving Artificial Potential Field. It was approved from the simulation result that local trap was eliminated effectively and vibrating was mitigated but not eliminated through adjusting the parameters of the functions. Referencing to man driving car, 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 [27] developed a technique for coordinating the paths of multiple robots 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. The planning is accomplished by first mapping the real space of the robots into configuration-space-time. Potential fields are applied around the C-Space-time obstacles and are used to modify the path of the robot. 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. In the method used to perform path planning here, a trial path through the C-Space-time is chosen and then modified under the influence of the potential fields until an appropriate path is found.

Marta C. Mora and Josep Tornero [8] 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.

Prahlad Vadakkepat, Kay Chen Tan and Wang Ming-Liang [28] proposed Evolutionary Artificial Potential Field (EAPF) for real-time robot path planning. 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 [24]. Potential field functions for obstacles and goal points are also defined. The potential field functions for obstacles contain tunable parameters. 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.

One popular obstacle avoidance method is based on edge detection called is 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 another 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 limits the accuracy in determining the spatial position of an edge to 10-50 cm, depending on the distance to the obstacle and the angle between the obstacle surface and the acoustic axis. Frequent misreading is caused by either ultrasonic noise from external sources or stray reflections from neigh boring sensors (i.e., crosstalk). Misreading cannot always be filtered out, and they cause the algorithm to falsely detect edges.

Secular's reflections occur when the angle between the wave front and the normal to a smooth surface is too large. In this case the surface reflects the incoming ultrasound waves away from the sensor, and the obstacle is either not detected or is "seen" as much smaller than it is in reality (since only part of the surface is detected). 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 paths.

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. Ultrasonic sensors, for example, have a conical field of view. A typical ultrasonic sensor returns a radial measure of the distance to the nearest object within the cone, yet does not specify the angular location of the object. 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 C 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 C_x 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. 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 C for obstacle representation. As in CMU's certainty grid concept, each cell (i, j) in the histogram grid holds a certainty value $c_{i,}$ 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. For ultrasonic sensors, this cell corresponds to the measured distance d and lies on the acoustic axis of the sensor. 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 neighboring cells are repeatedly incremented. This results in a histogramic probability distribution in which high certainty values are obtained in cells close to the actual location of the obstacle.

Next you apply the potential field idea to the histogram grid, so the probabilistic sensor information can be used efficiently to control the vehicle. As the vehicle moves, a window of w, x w, cells accompanies it, overlying a square region of C. We call this region the "active region" (denoted as C*), and cells that momentarily belong to the active region are called "active cells" (denoted as Cj). The size of the window is 33 X 33 cells (with a cell size of 10 cm x 10 cm), and the window is always centered about the robot's position. Note that a circular window would be geometrically more appropriate, but is computationally more expensive to handle than a square one. Each active cell exerts a virtual repulsive force Fi, toward the robot. The magnitude of this force is proportional to the certainty value c: and inversely proportional to d", where d is the distance between the cell and the center of the vehicle, and x is a positive real number (assume x = 2 in the following discussion).

At each iteration, all virtual repulsive forces are totaled to yield the resultant repulsive force Fr. Simultaneously, a virtual attractive force Fl of constant magnitude is applied to the vehicle, "pulling" it toward the target. The summation of F and Ft yields the resulting force vector R. In order to compute R, up to 33 x 33 = 1089 individual repulsive force vectors Fi, must be computed and accumulated. The computational heart of the VFF algorithm is therefore a specially developed algorithm for the fast computation and summation of the repulsive force vectors.

Combining concepts 1 and 2 (given above) in real-time enables sensor data to influence the steering control immediately. In practice, each range reading is recorded into the histogram grid as soon as it becomes available, and the subsequent calculation of R takes this data point into account. This feature gives the vehicle fast response to obstacles that appear suddenly, resulting in fast reflexive behavior imperative at high speeds [29].

But a problem occurs when traveling along the centerline between the two corridor walls; the robot's motion is stable. If, however, the robot strays slightly to either side of the centerline, it experiences a strong virtual repulsive force from the closer wall. This force usually "pushes" the robot across the centerline, and the process repeats with the other wall. Under certain conditions, this process results in oscillatory and unstable motion.

The Vector Field Histogram Method (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.

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 [30]. 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 [31]. 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.

In contrast, VFH* analyzes the consequences of heading towards each primary candidate direction before making a final choice for the new direction of motion. For each primary candidate direction, VFH* computes the new position and orientation that the robot would have after moving for a projected step distance d_s. At every projected position, VFH+ is again used to construct a new polar histogram based on the map information. This histogram is then analyzed for candidate directions, called projected candidate directions. By repeating this process n_g times, we build a search tree of depth n_g, where the end nodes (goals) correspond to a total projected distance of $d_t = n_g.d_s$.

The goal of this search process is to find a suitable projected trajectory of distance dt. Nodes in the search tree represent the projected positions and orientations of the mobile robot. Arcs represent the candidate directions leading from one position to another [31].

For every candidate direction c, a cost g(c) is calculated similar to the cost function used in VFH+. The cost associated with a node is simply the sum of the costs of the branches leading back to the start node. The primary candidate direction that leads to the end node with the smallest total cost is then selected as the new direction of heading d.

M. Yousef Ibrahim, L. McFetridge [28] developed the Agoraphilic algorithm which is a reactive local navigation technique based on the potential fields methodology. 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.

The dynamic window is especially designed to deal with the constraints imposed by limited velocities and accelerations, because it is derived directly from the motion dynamics of synchronic-drive mobile robots. This approach considers periodically only a short time interval when computing the next steering command to avoid the enormous complexity of the general motion planning problem. The approximation of trajectories during such a time interval by circular curvatures results in a two-dimensional search space of translation and rotational velocities. Search space is reduced to the admissible velocities allowing the robot to stop safely. Due to the limited accelerations of the motors a further restriction is imposed on the velocities: the robot only considers velocities that can be reached within the next time interval. These velocities form the dynamic window which is centered around the current velocities of the robot in the velocity space.

Among the admissible velocities within the dynamic window the combination of translational and rotational velocity is chosen by maximizing an objective function [34]. The objective function includes a measure of progress towards a goal location, the forward velocity of the robot, and the distance to the next obstacle on the trajectory. By combining these, the robot trades off its desire to move fast towards the goal and its desire to ship around obstacles (which decrease the free space). The combination of all objectives leads to a very robust and elegant collision avoidance strategy [34].

Nowadays fuzzy logic has become a means of collecting human knowledge and experience and dealing with uncertainties in the control process. Now fuzzy logic is becoming a very popular topic in control engineering [34]. 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 [34].

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, and C = (1-A*B) (the notation * is a t-norm operation in fuzzy set theory).

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 [9] developed a layered goal-oriented motion planning strategy using fuzzy logic is developed for a mobile robot navigating 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 favorable 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 [35] 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 omnidirectional 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 [32] developed a method to encode the fuzzy sets, fuzzy rules and procedure to perform fuzzy inference into expert system for behavior based robot navigation. The design of the behavior is based on regulatory control using fuzzy logic and the coordination and integration is defined by fuzzy rules, which define the context of applicability for each behavior. The complexity of robot behavior is reduced by breaking down robot behaviors into simple behaviors 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. Decision making process uses fuzzy logic for coordination, which provides a smooth transition between behaviors with a consequent smooth output response. In addition, the new behavior can be added or modified easily. Some of the experimental results are also shown for the Obstacle avoidance, Wall following and Goal-Seek behaviors.

The mentioned systems for *obstacles avoidance* and *goal reaching* are independent of each other. Recently the approaches is applied to navigation that is based an integrated model of both goal-acquisition as well as collision avoidance. This is challenging because the task of globa1 planning is far more computationally intensive than that of local navigation, and collision avoidance requires much quicker response times. Low [24] 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 to 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. Xiso et a1 describe an *Evolutionary Planner/Navigator* (EP/N) which in parallel recompute the optima 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. Its large capacity for online learning can potentially be extended to the dynamic scenario.

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 three dimensional a mobile robot navigation approach equipped with rule-based navigation algorithm which uses goal oriented approach to navigation and obstacle avoidance system is proposed. Mobile robot should navigated in dynamically changing environment and avoid from dynamic obstacle.

2.4 Summary

Basically, there are two kinds of path planning: one is global path planning which is based on entirely known environment; the other is local path planning which is based on real-time sensor information. In local path planning, the environment is unknown or partly unknown, which means that the obstacle's dimension, shape, position and number must be accessed by the sensors. In global path planning much calculation is needed to create the environment model, and need to locate the position of the robot in the environment continually. Besides, because of the uncertain factors in the environment, sometimes global path planning is unrealized. Up to the present, there come lots of ways to deal with the problem of path planning.

The main advantage of a mobile robot has to be useful for every place and it has to be cheap enough to build. Building a device which is bound to specific circumstances of an environment will not be useful although a device which is so expensive to build will be useful but you won’t be able to find people to use it. So while developing a mobile robot navigation algorithm main concern has to be its not being bound to environment (Dynamic) and it is being applicapable enough on cheap devices.

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. This makes it assume a priori knowledge of the workspace which make it a static environmental device and are usually based on the construction of the robot's configuration space which shrinks the robot to a point.

Even if the evolution on Artificial Potential Fields, which made it useful for dynamic environments, made it expensive. The Certainty Grid for Obstacle Representation again needs a highly capable ultrasonic sensor base. VFH, VFH+, VFH* methods are good but again they need more calculations than an easy microcontroller capable of.

Goal Oriented Navigation method and Behavior Fusion Methods attracted me because of their easy structure to build in industrial applications and their good enough results.

3. BEHAVIOR BASED DYNAMIC MOBILE ROBOT NAVIGATION

3.1 Overview

Making a robot to move to a specific position in a constantly changing environment with internal and external constraints is the greatest goal in a navigation system. To do so, the robot should recognize the environment around itself by using the navigation plan which can make the robot to accomplish the job using a set of sensors in the varying environment. When unexpected or un-modeled objects, static or moving, appear, the robot should have the ability of dynamically avoiding them, without any collision, and returning to the normal path after these objects have been passed or removed. Dynamic obstacle avoidance is also called local or reactive obstacle avoidance [1]. In contrast to static and global path planning, the dynamic obstacle avoidance approach only uses the local information of the surrounding environment. Consequently, it cannot always generate an optimal path, but it can make the robot react fast to unexpected obstacles. We can set a midway point outside the circle route and make the robot avoid the unnecessary route and reach the goal [1].

Traditional methods for mobile robot motion planning, referred to as model-based approaches, use a model of the environment to generate a path for the robot to follow.

Techniques for model-based path generation include road mapping, cell decomposition, and the potential field method. Model-based approaches could fail when dealing with the navigation problem of a real-world mobile robot since it is usually difficult or impossible to obtain an accurate model of a dynamic environment. For the mobile robot navigation problem in unknown environments, there exist sensor-based approaches which generate control commands based on sensory data. The main advantage of sensor based approaches is that the robot can navigate safely in a dynamic environment by reacting to obstacles detected by sensors in real time. A major drawback is that the robot may get lost even if a path to the goal exists.

A completely autonomous behavior based mobile robot is designed to operate in real world, which is usually complex and unstructured. The mobile robot must have the ability to respond quickly and appropriately to unexpected and adhoc situations [2]. On the basis of the stimulus-response Behavior in biosystem, Behavior based control has been proposed for robot navigation in unknown environments since this method does not need building an entire world model and complex reasoning process. A key issue in Behavior-based control, however, is how to coordinate conflicts and competitions among multiple reactive Behaviors efficiently. The coordination of multiple Behaviors is done by inhibiting those reactive Behaviors with lower levels [3]. In order to reach a specified target in a complex environment, the mobile robot at least needs the following reactive Behaviors:

1. Obstacle avoidance and decelerating at curved and narrow roads;

2. Following edges;

3. Target steer.

The components for providing autonomous capabilities of a mobile robot are grouped into few basic modules, namely motion planner, motion executor, motion assistant, and behavior arbitrator. The primitive motion executors such as obstacle avoidance, goal following, wall following for mobile robot navigation are developed in this pages Event driven concept is used on switching the motion behaviors. The resultant intelligence for mobile robot navigation is capable of efficiently performing motion behavior and detecting environmental event in parallel to adapt dynamically changed environment [3].

Various control architectures for development of autonomous mobile robot system defined the functionality of the three components in various ways, but the basic concept is similar. Briefly, the motion planning is responsible to plan the primitive motion behaviors with a key issue of behavior-based control is how to efficiently coordinate conflicts and competitions among different reactive behaviors to achieve a good performance.

3.2 The Navigation System and Landmark

The objective this thesis is to develop navigation system for mobile robot capable of traversing predefined path while avoiding obstacles. For its ideal implementation the mobile robot control system must be capable of employing multiple sensors and efficient algorithms. By analyzing local frame detecting and avoiding obstacles the robot will make decision about the direction to the goal. (Figure 3.1) Fig.3.1 demonstrates an example of scene of mobile robot navigation. Here obstacles are shown by rectangles, goal by "+". Robot has five sensors which are used for detecting obstacles. The problem is reaching the goal by avoiding the obstacles.

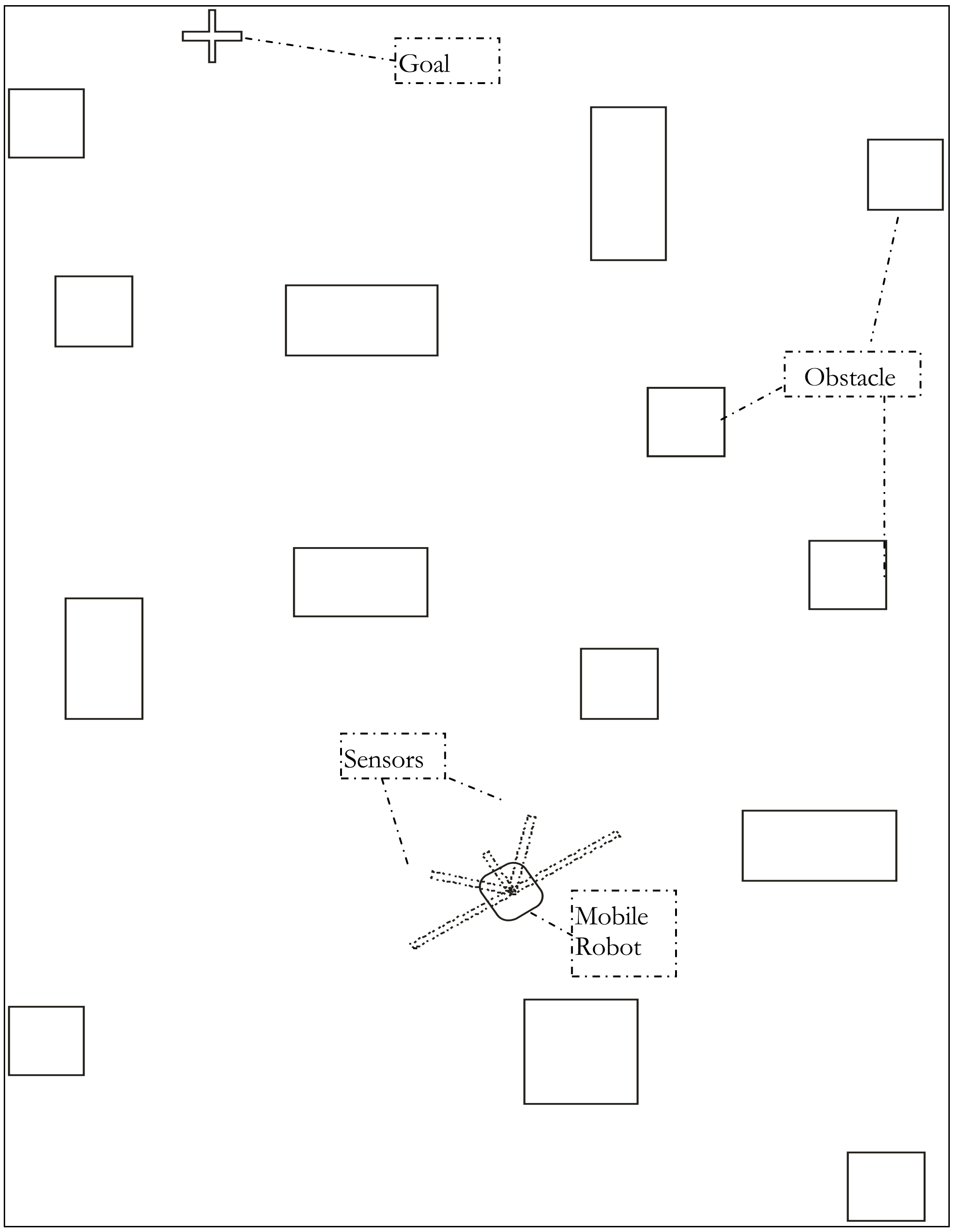


Figure 3.1 Navigation Scene

The navigation system is comprised of a set of modules, each module connected to other modules via inputs and outputs. The system is closed, meaning that the input of every module is the output of one or more modules. It uses all available local information in real time to identify the right sequence of local actions for the robot. The navigation system find desired direction and adapt to the obstacles. Some obstacles are dynamic; the environment is characterized with fast-changing dynamic and moving obstacles. Navigation system based on information that receives from the sensors, move to the goal avoiding obstacles. It combines the versatility of responding to environmental changes and new goals with the fast response of a tactical executive tier and behaviors. The algorithm proposed in presents a novel navigation scheme that decomposes the 3-D navigation problem into several 2-D navigation problems. Each 2-D navigation system uses a preference-based fuzzy behavioral controller that has a general and cooperative fusion mechanism instead of using a weighted sum of the schema outputs.

3.3 Object Detection and Representation

The system designed in the thesis is mainly based on threshold distance bounded. Five range finding sensors are used for range detection of the environment. Each sensor is set on a specific location around the robot and the thresholds of activation for the sensors are shown by the arrows in Figure 3.3. The angle from Y-axis to the center of beam is 0^o sensor and obviously right front one is 45^o sensor and left front one is -45^o sensor. These sensors have two thresholds one of them is Long Threshold which equals to R and the other threshold is short which equals to d. Suppose the robot radius is r which is defined by from the far point to the center of the robot and R-r is the needed distance to slowdown from the fastest speed ($V(\max)$) to threshold activated speed ($V(safe)$) which is the speed while we are using on avoiding obstacles. The safe range between robot and obstacle is d which cannot be lower than needed distance from threshold activated speed ($V(safe)$) to zero $(V = 0)$. And we need to add the sensors' error distance ($\acute{e}$) and distance that will pass through detection and the reaction time.

And if we think that other kinds of robots are in the system. Then our threshold ranges needed to be multiplied by 2 to avoid collision the robots in the system. 90^o and -90^o

sensors thresholds must be longer than $R.cos(45)$ which will be mentioned in Obstacle Avoiding section. The explicit rules can be written as:

$$R > 2.(r + V(max).t_1 - \frac{1}{2}a_s.t_1^2 - V(safe).t_2 + \frac{1}{2}a_S t_2^2 + V(max).t_3 + \acute{e}) \quad (1)$$

$$d > 2.(V(safe).t_2 - \frac{1}{2}a_S t_2^2 + V(safe).t_3 + \acute{e}) \quad (2)$$

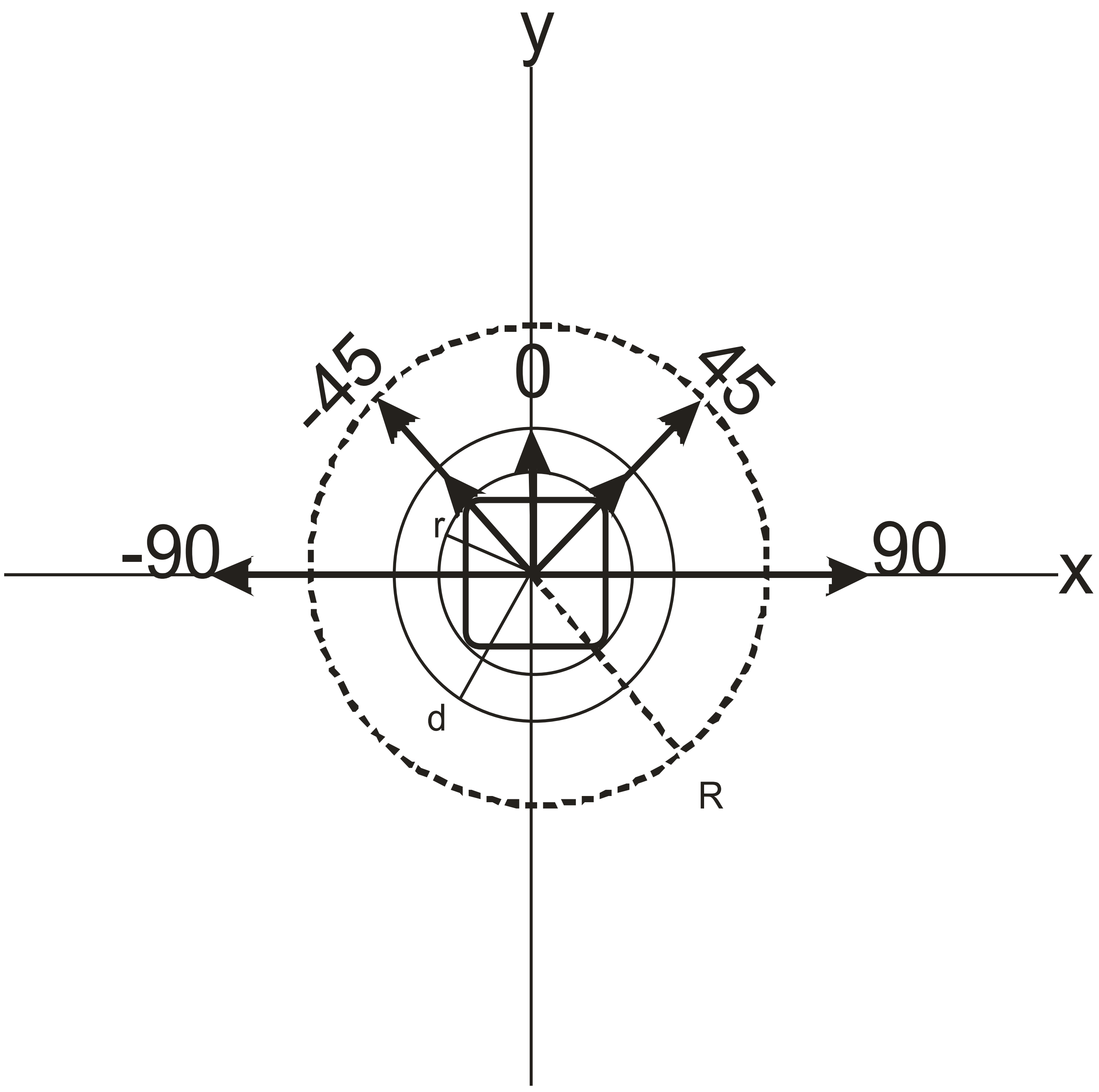


Figure 3.2 Sensor Range of Robot

Where;

t_1: Deceleration time from V (max) to V (safe);

t_2: Deceleration time from V (max) to V=0;

t_3: Time period for detection and reaction.

3.4 Behavior Definitions

The motivation toward behavioral robotics comes from the need to represent the reactive abilities of humans, animals insects etc. to sensed phenomenon. The way moths are attracted to light and the way in which ants follow each other in long trails toward food are two simple examples of this, whereas the instinctive reaction of a person to avoid an oncoming car, or the flight or fight response of creatures in dangerous positions are complex displays of behavioral (reactive) control at work. Behavioral control attempts to solve the problem of self-automated control systems needing to know everything about the environment they work in. This can often clutter systems with too much knowledge and make real time control a slow and difficult process. The whole idea behind this philosophy is that humans (and animals) react in their environment by sensing (often unpredictable) information and yet we respond quickly with reactions to survive and operate in a changing and potentially unknown world. It is this method of operation that has shown that real time control can be based on sensing the environment and following a simple set of rules to exact an action or a set of actions.

Definition for the *robot behavior* is given by Mataric's, "A behavior is a control law that satisfies a set of constraints to achieve and maintain a particular goal" [37]. The behaviors used for the control of the robot are defined. Each primitive behavior is self-contained and reacts to data from specific sensors. Basically in Figure 3.2 there are two main behavior addicted to robots which are *Goal-Seeking* behavior and *Obstacle Avoidance* behavior. If robot does not have any signals from obstacles then it uses goal seeking behavior. Obstacle avoidance behavior is built by some sub behaviors like *Forward Move*, *Backward Move* and *Fallow-Wall* behavior. For instance, *Forward Move* behavior reacts from 45^0 and -45^0 sensors, *Follow-Wall* behavior reacts from 90^0 and -90^0 sensors, and *Backward Move* behavior reacts from 0^0 $45S^0$ $-45S^0$ sensors. The architecture of the behavior-based logic of the mobile robot is shown in Fig. 3.2 Using the information received from the *sensors* on every frame program applies a force on the moving robot. All behaviors are implemented using Rule Bases.

The *Very-Close* behavior is an "internal behavior" used to invalidate commands that could lead to collisions in situations when the robot is very close to an obstacle. In order to reach its goal, *Reaching Two-Dimensional (3-D) location (XY)*, program will continuously switch to the behaviors that are currently activated by the sensory information. For example, if none of the sensors detects something, the *Goal Seeking* behavior will be selected. If there is information coming from 45^0 and -45^0 sensors sum of reactive forces of Forward Move will set Forward Behavior. If the program receives only information from the sensors on one of its sides (90^0 and -90^0), which means there is an obstacle on one side, the *Fallow-Wall* behavior will be selected. If the program receives information from the sensors $45S^0$ and 0^0, which means there is an obstacle front right of it, the sum of forces will shape the *Backward Move* behavior.

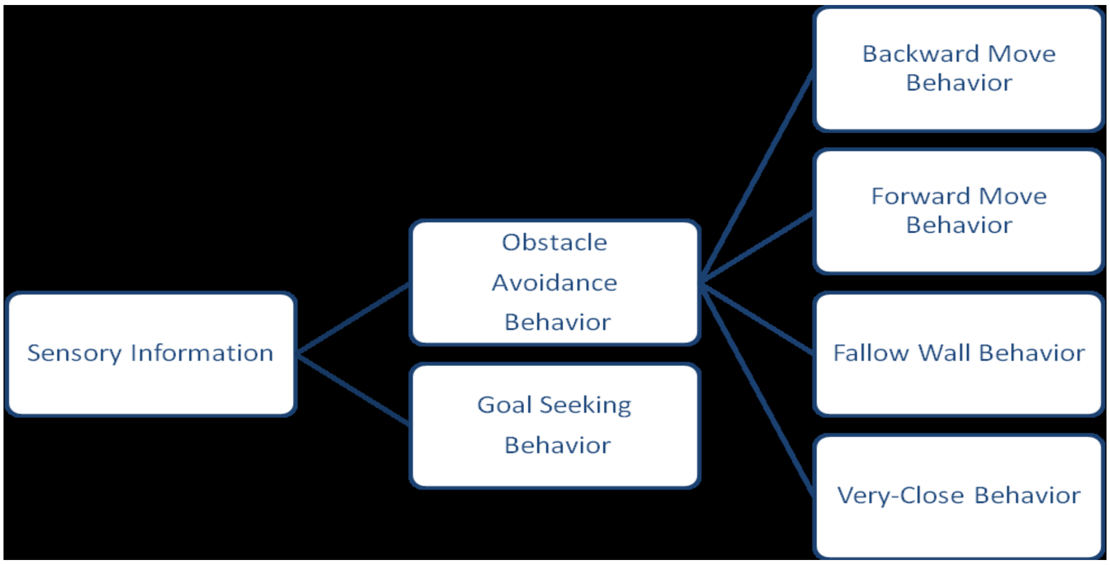


Figure 3.3 Behavior Definitions

3.5 Goal Seeking Behavior and Obstacle Avoidance

Goal seeking behavior is, if no obstacles detected, the mobile robot will concentrate on moving to the goal. When the environment doesn't favor the action of goal attraction, however, other behaviors will be activated. This behavior needs to get the information from sensors continuously. When the sensor data is over the threshold value, the goal seeking behavior will take into action.

The objective of the goal seeking behavior is to steer the robot approaching a given goal position straightly. Suppose the coordinate of goal is (Xg, Yg) and the coordinate (X (t), Y (t)) for robot at time with respective to the start position is given. The direction from robot at time t to the goal position can be written as θ (t), where

$$\theta(t) = \tan^{-1}\frac{Y_g - Y(t)}{X_g - X(t)} \qquad (3)$$

When we found the $\theta(t)$ the robot will turn to its objective. After determination θ angle robot follows to the goal. During movement of robot, in order to avoid from obstacle, use rule base. The rule base uses input signal obtained from the sensors of the robot.

In design (Figure 3.4) every sensor has a force vector for example 90 and -90 degree sensors has an equal weighted (the force have to be applied on the robot) unit vector which tells the robot to go forward which can be considered like a Force from the centre of the robot.

Robot's sensors have a hierarchy between them the 0^o sensor is on the top of this hierarchy which means that if 0^o sensors detect something our robot have to go backwards because there is an obstacle in front of it. If obstacle detected sensors have the same hierarchy of action then we have to calculate the total force to apply the robot;

$$F(t) = F(45^\circ) + F(-45^\circ) = F(1,1,0) + F(-1,1,0) = F(0,2,0) \qquad (4)$$

Here F is the force to be applied on the robots local frame. $F(45^o)$ is the reactive force have to be applied on the local frame when 45^0 long sensor triggered which equals F(1,1,0) in X,Y,Z frames. In F(1,1,0), the first value is the Force need to be applied in X frame, the second is the Force need to be applied in Y frame, and the third value is the Force need to be applied in Z frame.

Making robot angular movements, a side force have to be applied from the frontier position of it on its local frame. For example the definition of F (45), this equals to F (1, 1, 0) from the rule table,

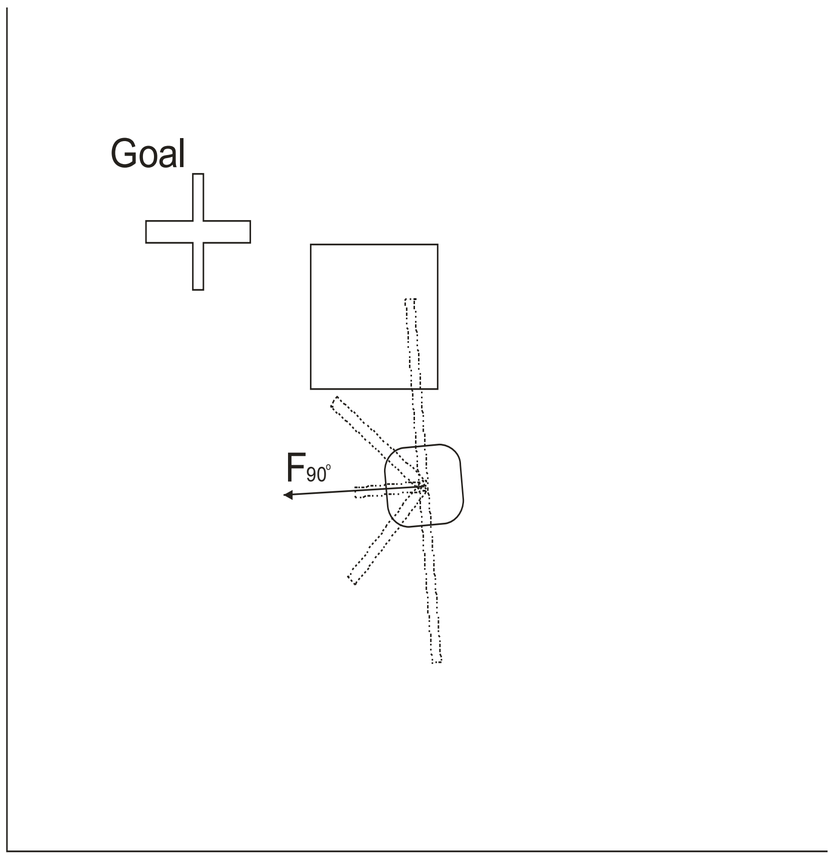


Figure 3.4 90º Force Applied Robot

F (45) = F (1, 1, 0) =

LocalForceAtLocalPosition (V (0, 1, 0), V (1, 0, 0)) +

LocalForceAtLocalPosition (V (0, 0, 0), V (0, 1, 0)) (5)

Here the first component define the position of the force on robot which is based on its local frame and second component define the force vector have to be applied on the local frame of the robot. In real robotic applications the capabilities of robot can vary from the design or the capabilities of the robot's components so weight of force can change up to capabilities of the robot's motor or the mass of the robot.

Below Table 3.1 gives the rules to apply forces over robot. These rules are being found by experimental studies over the simulation. Weight of force which represented in the table as "k" can be changed.

Table 3.1 Move Rule Table

Hierarchy	Sensor	Force
1	0	$F(1k, -2k, 0)$
1	-45S	$F(-2k, -3k, 0)$
1	45S	$F(2k, -3k, 0)$
2	-45L	$F(-1k, 1k, 0)$
2	45L	$F(1k, 1k, 0)$
3	-90 90	$F(0, 3k, 0)$

The calculation gives only the direction for us. First our robot acts to turn that direction with the weighted force and it calculate that on every step and this keeps on running until all the sensors off. The X(t) component of the force affects the angular velocity for the robot and the Y(t) component effect the linear velocity for the robot which means that you can't apply the force same position of the robot;

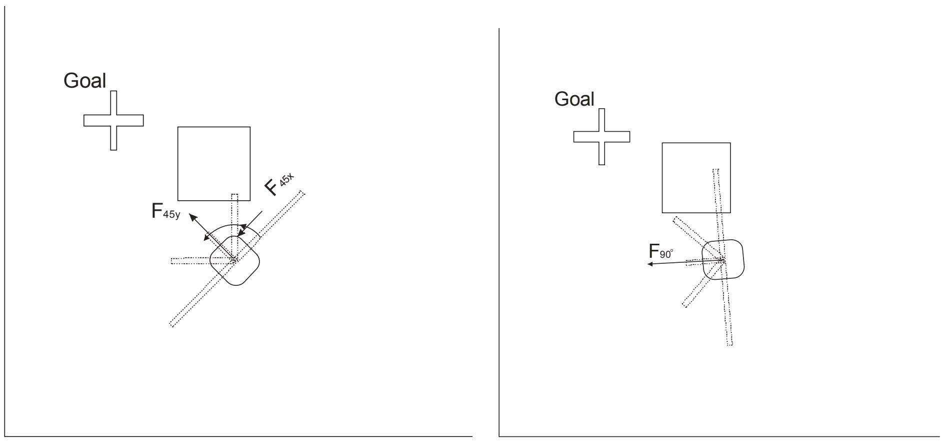


Figure 3.5 Linear Effects and Angular Effects

It is understandable from the Figure 3.5 if there is a wall by the sides of the obstacle it automatically try to follow the wall which can be expressed like wall fallowing manner. As mentioned above -90 and +90 sensors have to be longer than the R.cos (45^o) if we make shorter than R.cos (45) the wall fallowing manner will not work.

In mobile robot the lengths of detection of obstacle of (-45^o) and (45^o) sensors 1.5 time are longer than the lengths of detection of obstacle of (90^o) sensor. And also the lengths of detection of obstacle of left and right sensors 1.5 times are longer than the lengths of detection of obstacle of (90^o) sensor. If our robot head through an obstacle when it is parallel to Y=0 on the local frame of mobile robots first it will try to go forward after it detects if the robot don't make any angular movement and just go back our robot will always try to act this manner. Need for change the parallelism with the obstacles face and the X axis of our robot which will cause our robot's -45L or 45L sensor to detect it and turn. Figures 3.6-3.7 demonstrate different scene of robot with obstacle.

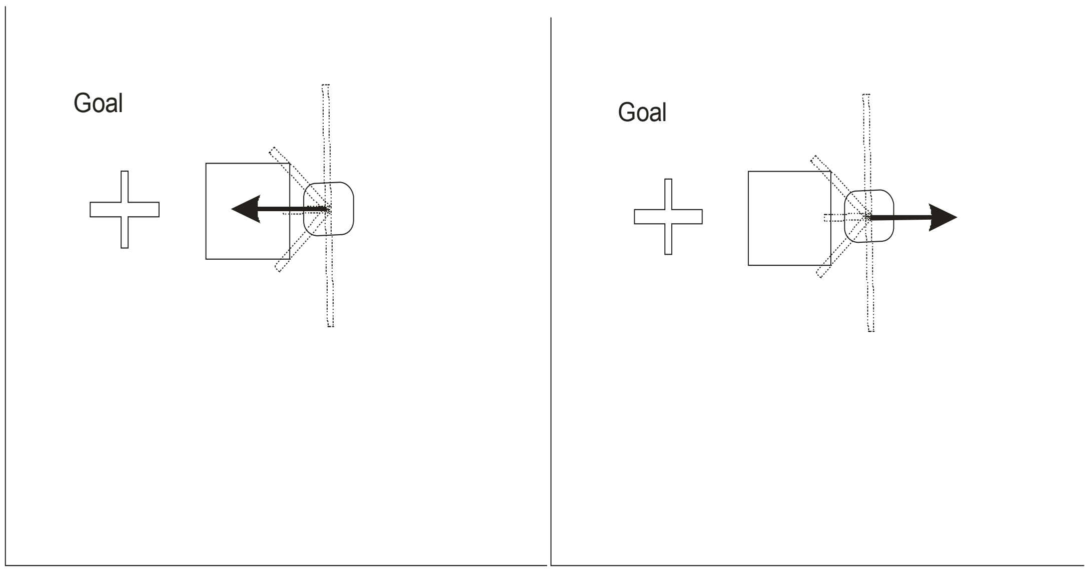


Figure 3.6 Axis Parallel

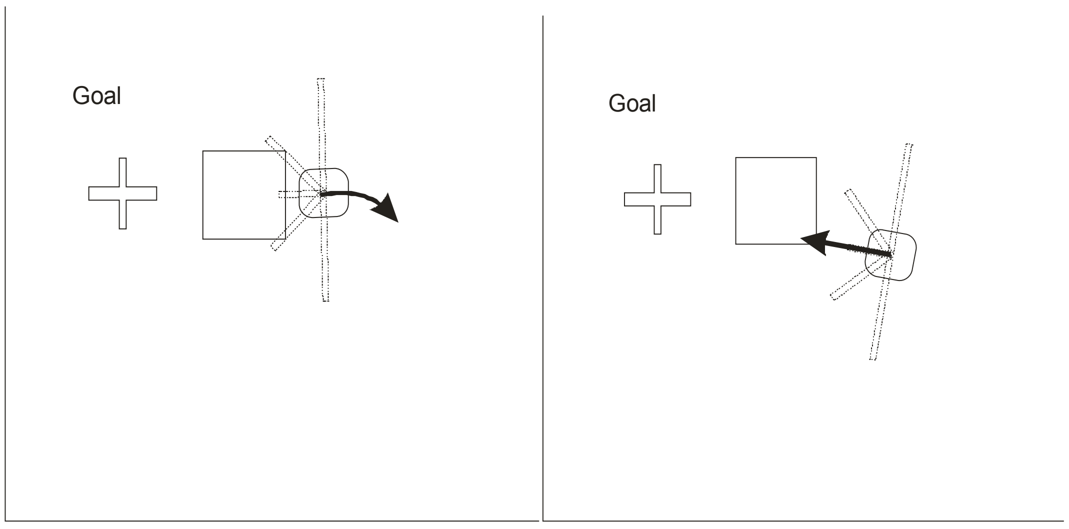


Figure 3.7 Axis Imparallel

After the obstacle avoided the robot enters the goal fallowing manner and starts to turn to obstacle. Because of wall fallowing manner which will cause a curvy manner.

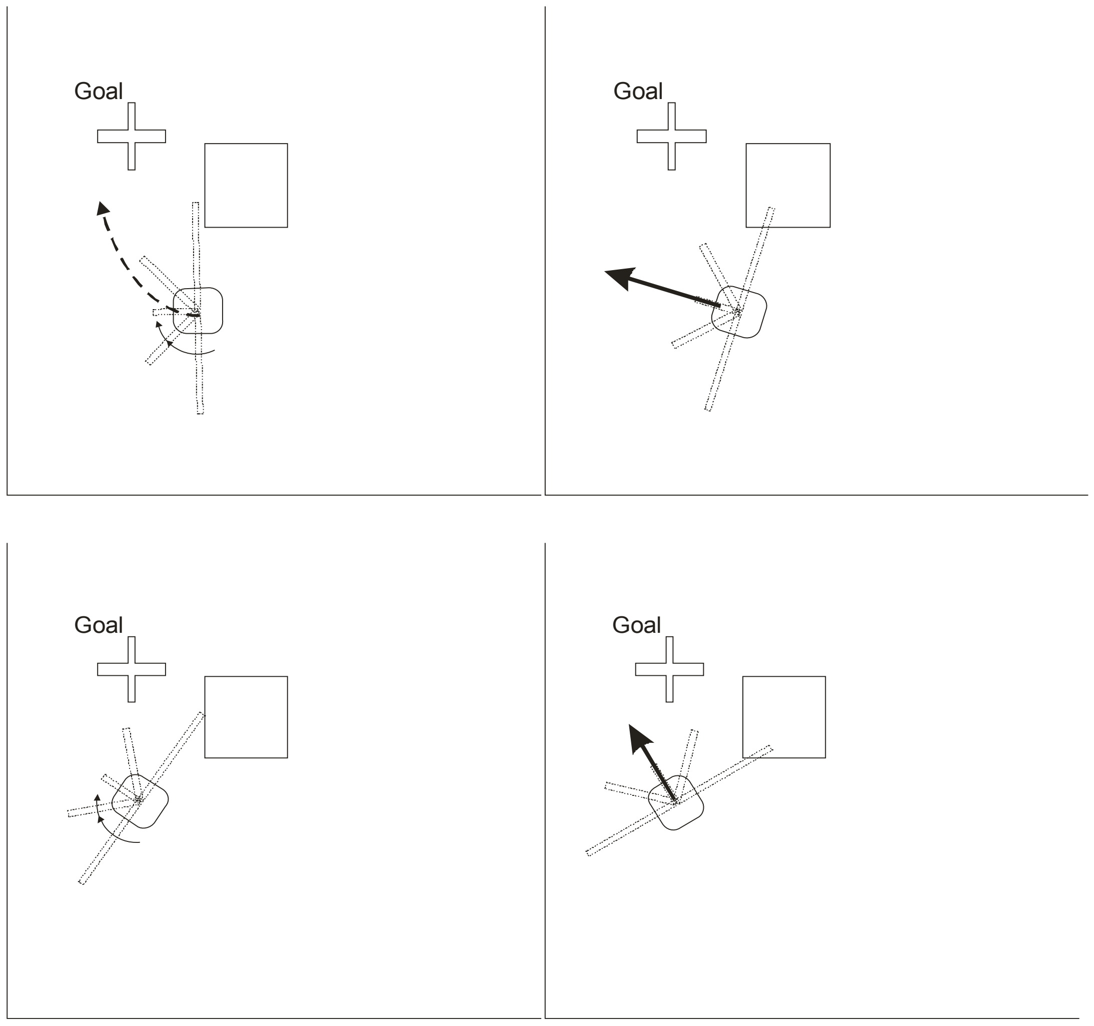


Figure 3.8 Construction of Rule Base for Obstacle Avoidance

As mentioned robot has five distance sensor when measured between it and the obstacles except -90 and 90 sensors each sensor has two (minimum and maximum) threshold distances. Using these minimum maximum distances the rule base is constructed for obstacle avoidance. These rules are obtained from different scene and they allow to implement behaviors depend on values of sensor signals. Some scenes are shown below states. Fragment of rule base is given below;

If d (0) > T_1 and d (45L) > T_1 and d (-45L)>T_1 and d (-45S)>T_1 and d (90)>T_1 and T_2 < d (-90) < T_1 then GO FORWARD θ=0

If d (0) > T_1 and d (45) > T_1 and T_2 < d (-45) < T_1 and d (90)>T_1 and T_2 < d (-90) < T_1 then θ = "TURN RIGHT UNTIL d (-45) $\geq T_1$"

If d (0) > T_1 and T_2 < d (45) < T_1 and T_1 < d (-45) < T_2 and d (90) > T_1 and d (-90) > T_1 then GO FORWARD θ=0

If d (0) > T_1 and T_2 < d (45) < T_1 and d (-45)>T_1 and d (90)>T_1 and T_2 < d (-90) < T_1 then θ= θ = "TURN LEFT UNTIL d (-45) $\geq T_1$"

If d (0) < T_2 and d (45) < T_2 and d (-45) < T_2 and d (90) > T_1 and T_2 < d (-90) < T_1 then GO FORWARD θ=0

Surveys of States:

Obstacle avoidance behavior and goal seeking behavior are the two most important behaviors that make up the functionality of mobile robot. Few scenarios with different arrangement of obstacles and goal will be used to test the capability of mobile robot.

State-1

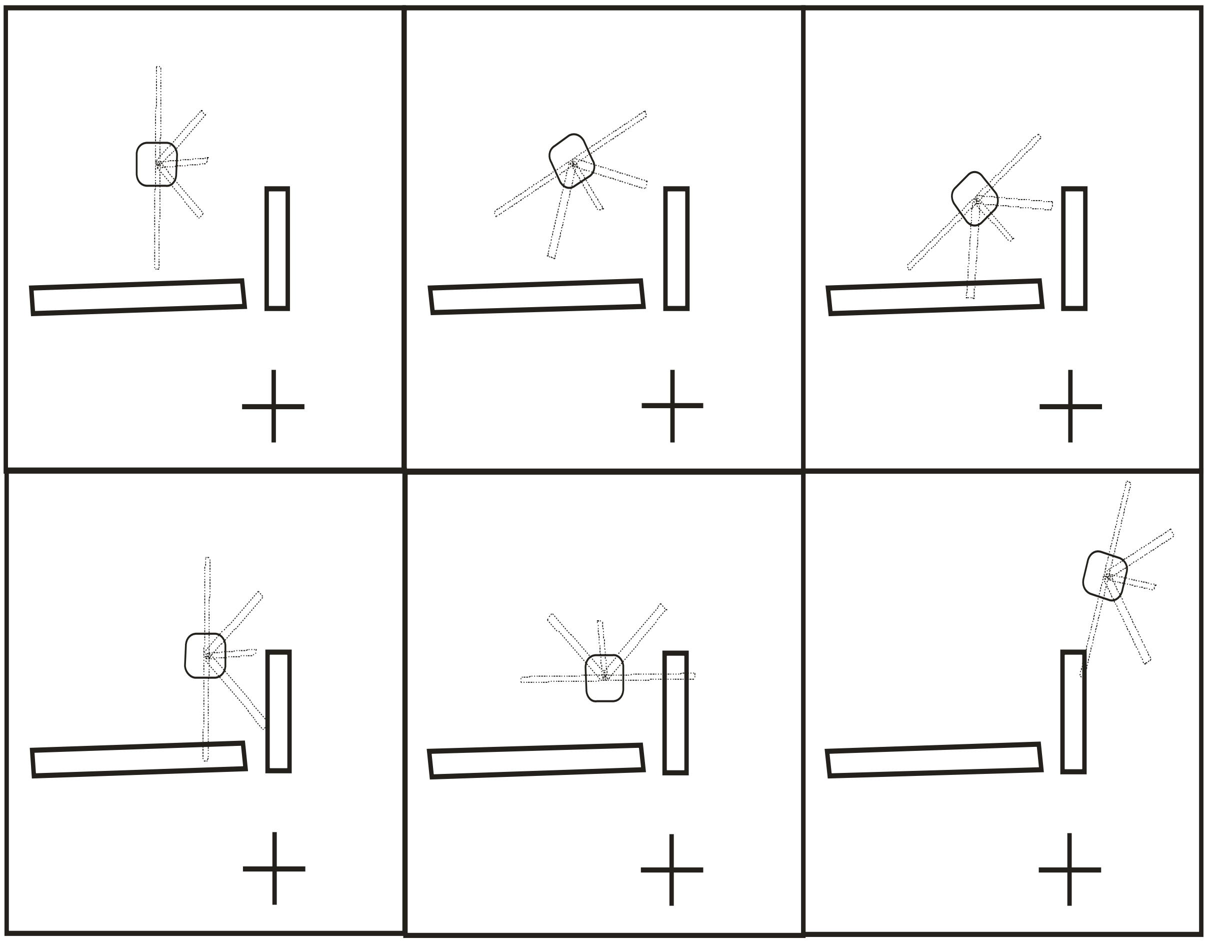


Figure 3.9 State-1 Obstacle Avoidance

In State-1 as a beginning action, because of no obstacles detected, our robot will try to turn the goal and go forward. First the 45^o sensor detects an obstacle and it will turn left. Because of the hierarchy 90 ^{o} sensor will not be activated until 45^o sensor leaves the obstacle. When 90 ^{o} sensor activated, it will start to go forward and then again 45 ^{o} sensor activates because of the hierarchy it will start to turn left again

State-2

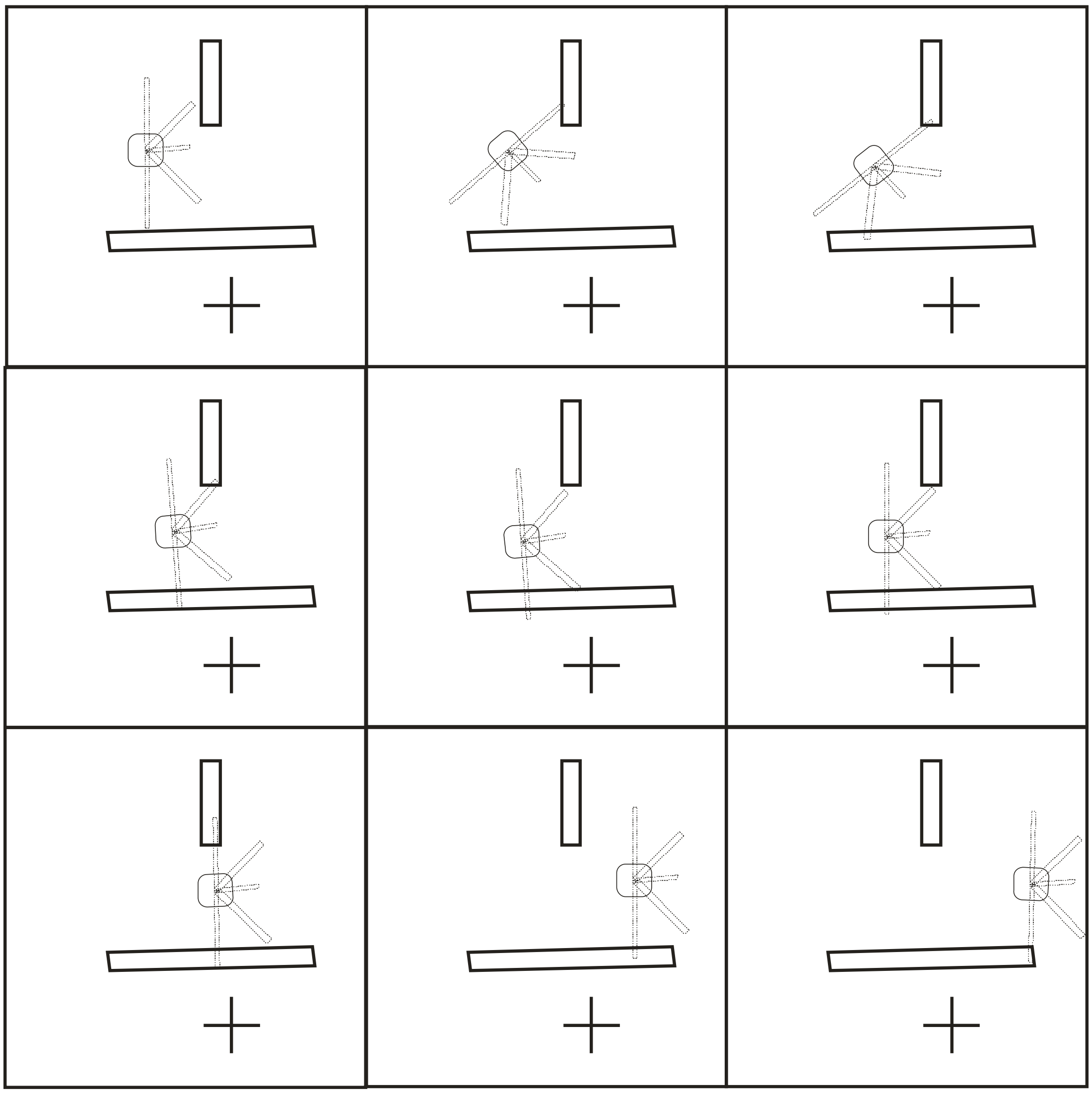


Figure 3.10 State-2 Obstacle Avoidance

In State-2 as a beginning action robot will try to turn through the goal point while turning -90 degree sensor triggers and it starts to go forward. Then 45 degree sensor will be triggered because of hierarchy and our robot start to turn left when 45 degree sensor off our robot will start to go forward again because of 90 degree sensor on his way if -45 or 45 degree sensors triggers while they are trying to turn obstacles will make them both triggered because of their same hierarchy the sum of actions will cause the robot go forward after they both leave the obstacle there is only 90 degree sensor stays triggered which will cause a wall fallowing manner for the robot. After it leaves the obstacle it enters the goal seeking behavior.

State-3

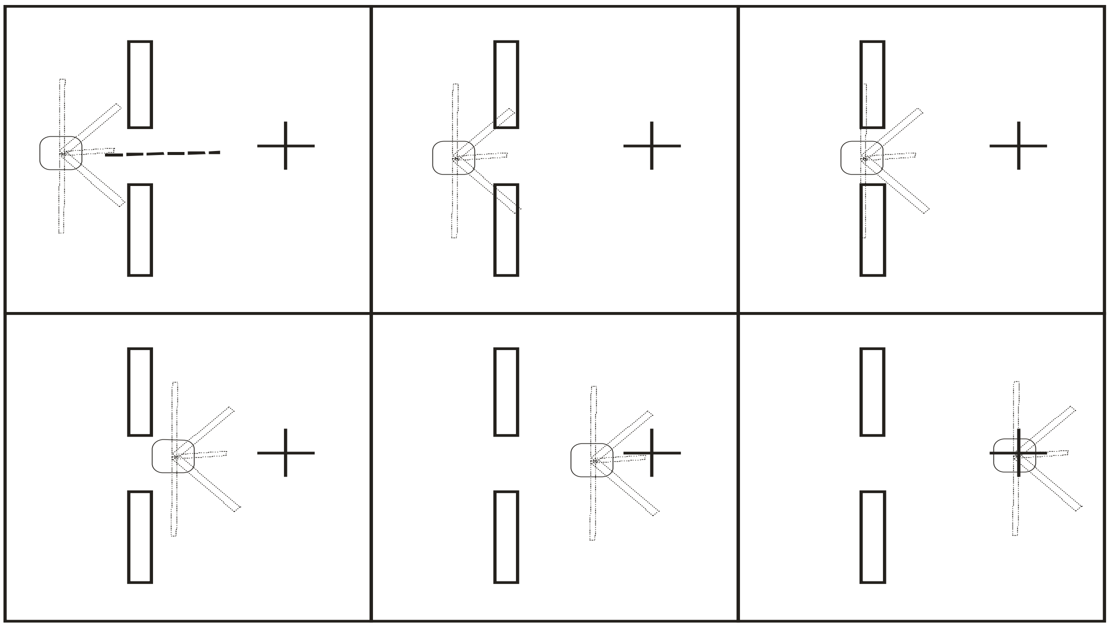


Figure 3.11 State-3 Obstacle Avoidance

In state-3 while our robot heading through goal, robot's both 45^o and -45^o sensors are triggered. As a reason of hierarchy we take the sum of actions what makes our robot to go forward. After sensors down it will enter the goal seeking behavior again.

In state-3 there is another situation confronting us what if our 45^o sensors activate backward movements (d (45^o) < d). It will behave in Figure 3.12;

In State-4 (Fig 3.13) there is an obstacle right in front of our robot if sensors both triggers at the same time, even it's very hard to happen, robot will go forward. After 0^o sensor trigger it will go back right and when it tries to go through the goal again 45^o sensor will trigger again, which make it turn right. After 45^o sensor down 90 ^{o} sensor will be active and that will make our robot enter wall fallowing manner. After all sensors down our robot will enter goal seeking manner and reach the goal point.

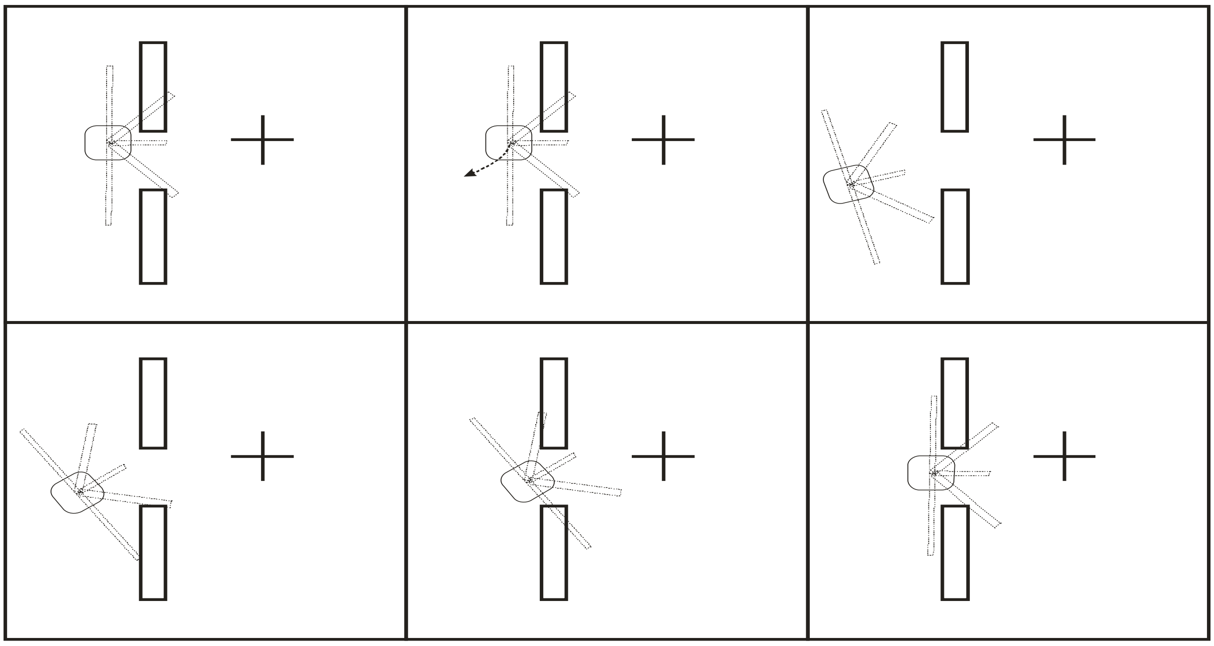


Figure 3.12 State-3(b) Obstacle Avoidance

State-4

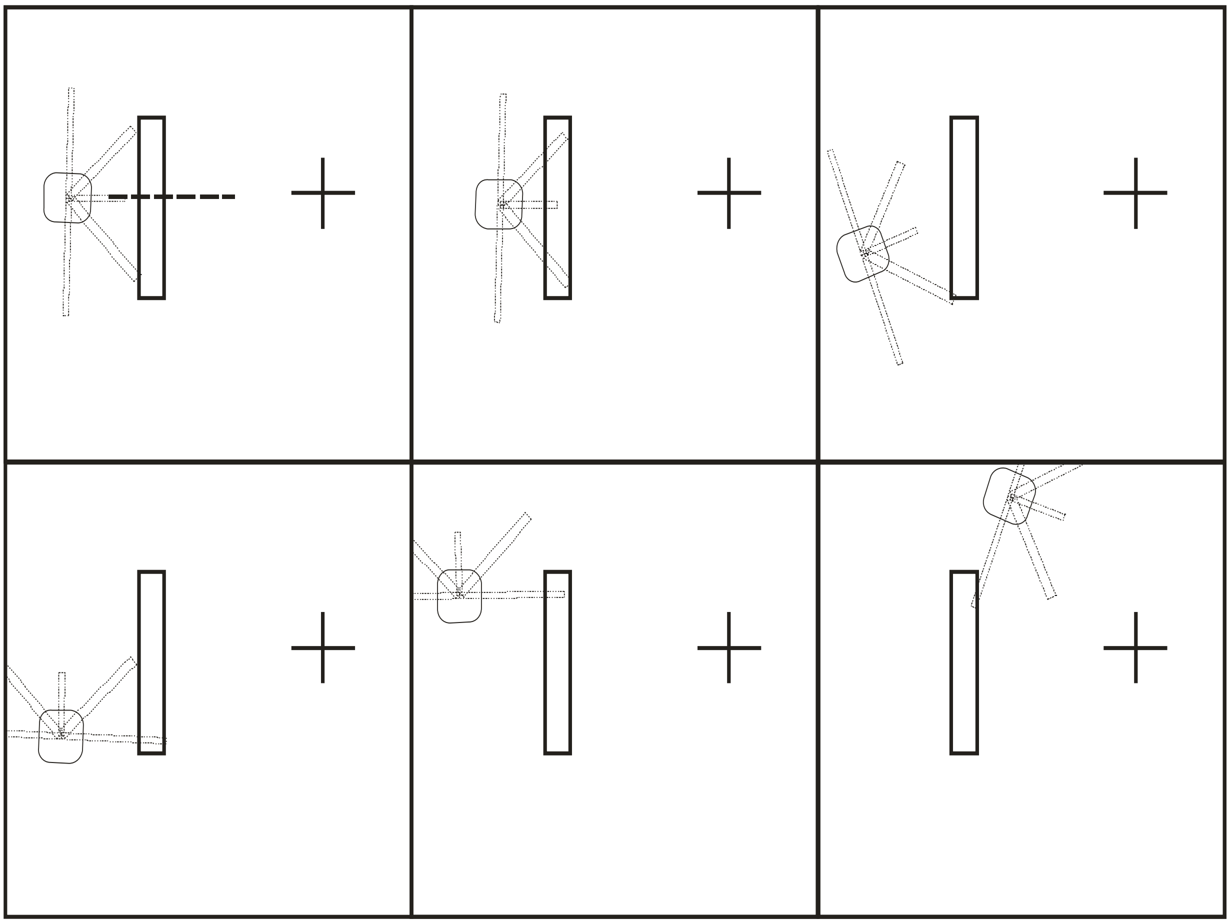


Figure 3.13 State-4 Obstacle Avoidance

State-5

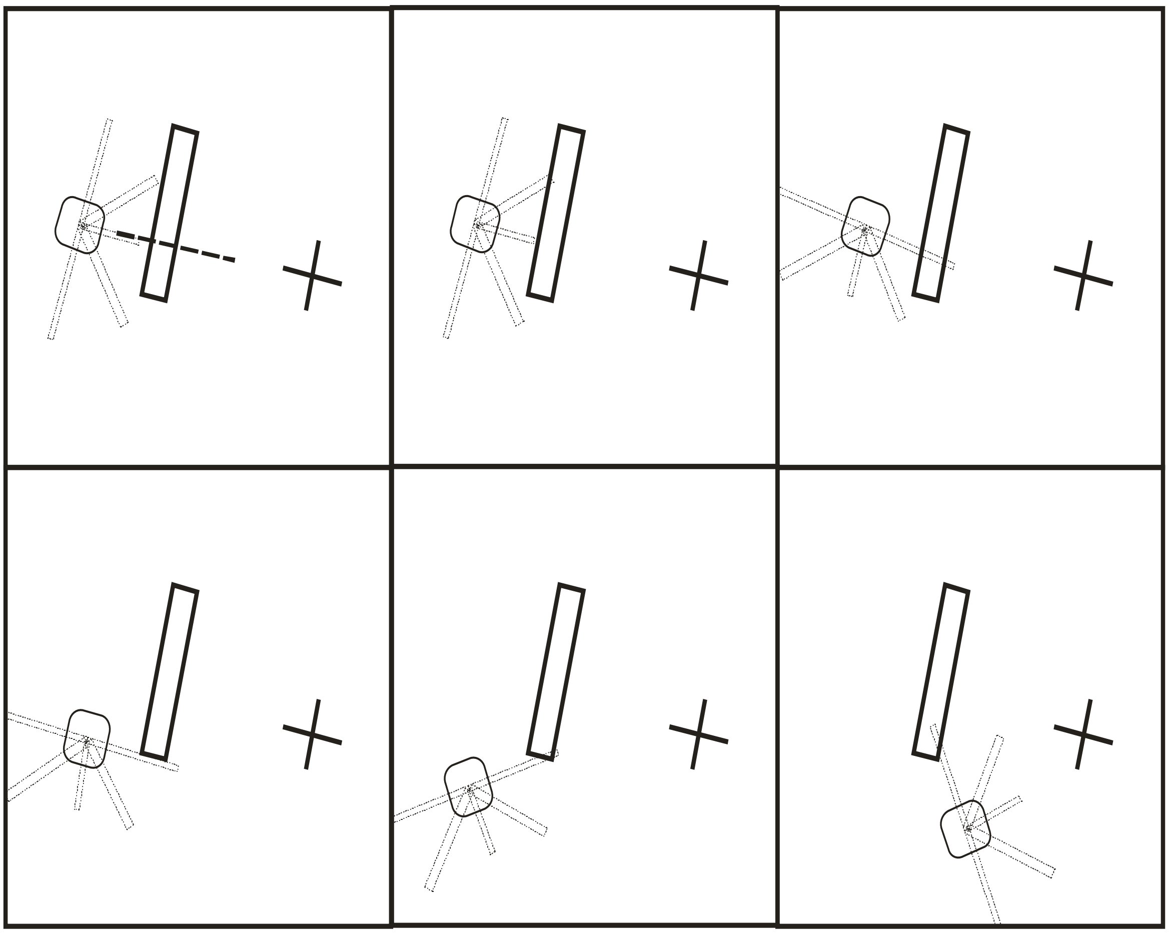


Figure 3.14 State-5 Obstacle Avoidance

In State 5 again there is an obstacle in front of robot but the difference between Stage 4 only -45^o sensor first detects the obstacle as a result of that our robot will turn right until -45^o sensor of which will cause -90^o sensor activated and our robot will enter the wall fallowing manner until the robot leaves the obstacle.

3.6 Summary

In the reactive approach to autonomous navigation, information about obstacles and goal position is used simultaneously as global information, which may result in a shortsighted behavior in some situations. Hierarchy is used to fix the priority of all the behaviors of the mobile robot. After receiving the data from the sensor, computation and calculation will be done to decide the action. The behavior rules adjust the degree of attention depending on environment, activating and task limiting conditions.

There are 5 sensors used for building this system which are located with 45^o differences. And the threshold distances are built by the speed factor and the rules. As a result of collision free acting obstacle avoidance behavior has a higher priority than the goal seeking behavior which is solely depends on hierarchically bind sensors. After obstacle's distance and sensor information gathered if obstacle avoidance is activated, the mobile robot will calculate the action based on this information.

4. IMPLEMENTATION AND EXPERIMENTAL RESULTS OF MOBILE ROBOT NAVIGATION

4.1 Overview

To build a real time dynamic obstacle avoidance simulation first of all we need to build the world that we are going to work on and the world has to be as real as possible, which means while building the navigation system of robot, as a rigid body, have to confront the real world's physical problems like friction, damping, acceleration, momentum etc., which is called kinematics of a rigid body. And we need to visualize some situations like collision, acceleration, and for our design sensor detection.

The different libraries are used for development of software. In 3D Game Programming physics engines are used to simulate the real world. In this thesis NVIDIA PhysX Engine SDK is used to build the effects of the real world [38]. The rigid body dynamics component enables you to simulate objects with a high degree of realism. It makes use of physics concepts such as reference frames, position, velocity, acceleration, momentum, forces, rotational motion, energy, friction, impulse, collisions, constraints, and so on in order to give you a construction kit with which you can build many types of mechanical devices.

The SDK has an ANSI C++ programming interface. Internally the SDK is implemented as a hierarchy of classes. Each class that contains functionality that can be accessed by the user implements an interface. This interface is effectively a C++ abstract base class. In addition, some stateless utility functions are exported [38].

In order to provide a certain degree of portability, the SDK makes use of size specific type definitions. The SDK's classes NxVec3, NxMat33, NxMat34, and NxQuat represent a 3-vector, a 3x3 matrix, a 3x4 matrix, and a quaternion respectively. They are currently configured to use single precision (32 bit) floating point scalars. These are the types the user should use when interfacing with the SDK [38].

4.2 How to Build a Robot

Actors are the protagonists of a simulation. Actors have two basic roles: static objects, fixed in the world reference frame, which are the obstacles, or dynamic rigid bodies, what is a robot, in the simulation [38]. One important aspect of actors is that they can have shapes assigned to them. Collision detection ensures that the shape of one actor does not intersect with the shape of another actor. Shapes can also be used to trigger various behaviors when another shape intersects them.

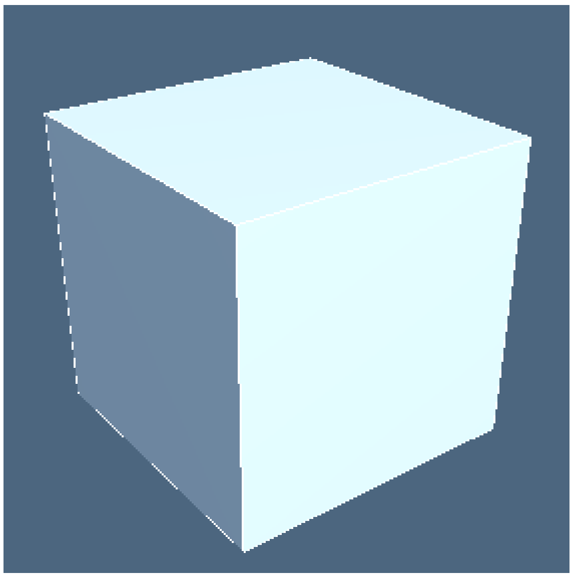


Figure 4.1 Cube

According to the laws of physics, any rigid object of any shape may be perfectly represented by an inertia tensor and by a point mass located at the object's center of mass. This is the approach used by the NVIDIA PhysX SDK and most other rigid body dynamics libraries. The SDK will optionally compute the inertia properties of the bodies using the assigned shapes, or the user can supply custom parameters [38].

As a robot we used a 1*1*1 cube of a dynamic actor to simulate the behaviors of the robot. This actor as a rigid body is effected by forces like directional forces built by directional forces and environmental forces like friction and damping.

Our create cube function both work with the static or dynamic objects and all the objects except goal point and plane actors built with this function.

Also it is possible to build more than one dynamic robots to represent the system with a more realistic way.

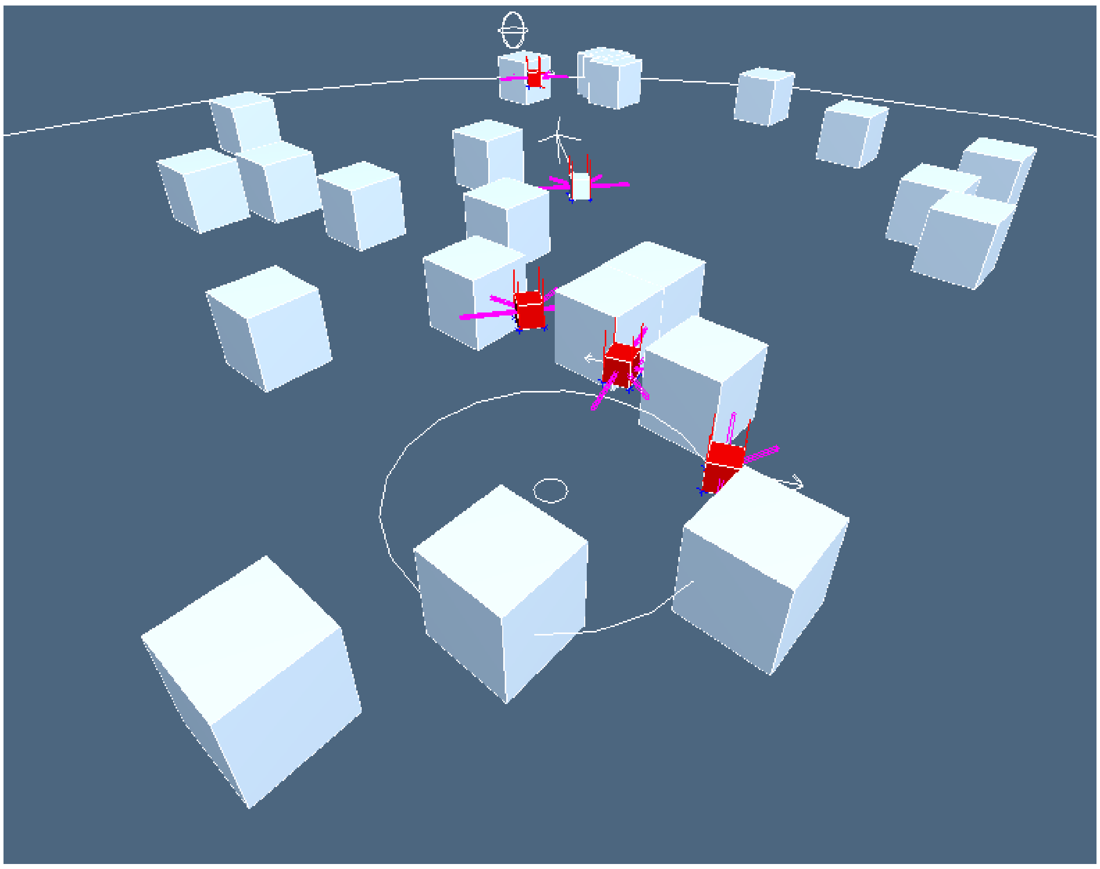


Figure 4.2 Multiple Robots

4.3 How to Build an Artificial Sensor

If you are building an artificial robot you have to think every aspect of it. It has to act as if it is in real world otherwise your results can't show the real calculations.

First of all, the sensor must act as if they are in real world. It needs to have a threshold distance to react if there is an obstacle closes enough to mention. Sensors needs to have dimensions for reaction and robot mustn't know the world far-reaching than robot's sensor reaches. I build that based on the AI Sensor Sample in Engine Samples.

I used trigger report which is a support of engine. A trigger is a shape that permits other shapes to pass through it. Each shape passing through it can create an event for the user when it enters, leaves, or simply stays inside the shape trigger send an event.

To receive trigger events, the SDK must be told where to send them. Trigger events are passed to a user defined object of type NxUserTriggerReport.

C++ application of it;

```
class SensorReport : public NxUserTriggerReport
{
  virtual void onTrigger(NxShape& triggerShape, NxShape& otherShape,
NxTriggerFlag status)
  {
    NX_ASSERT(triggerShape.getFlag(NX_TRIGGER_ENABLE));
    if(status & NX_TRIGGER_ON_ENTER)
    {
      NxActor& actor = otherShape.getActor();
      if(actor.userData)
      {
        NxActor& triggerActor = triggerShape.getActor();
        if(triggerActor.isDynamic()
                                && triggerActor.userData != NULL)
        {
          MyObject* Object = (MyObject*)triggerActor.userData;
          Object->nbTouched++;
          Object->setForceType((char*)triggerShape.getName());

        }
      }
    }
    if(status & NX_TRIGGER_ON_LEAVE)
    {
      NxActor& actor = otherShape.getActor();
      if(actor.userData)
      {
        NxActor& triggerActor = triggerShape.getActor();
        if(triggerActor.isDynamic()
                          && triggerActor.userData != NULL)
        {
          MyObject* Object = (MyObject*)triggerActor.userData;
          Object->nbTouched--;
        }
      }
    }
  }
}
```

Figure 4.3 Fragment of Sensor Report Program

Here MyObject class gives us information about the triggered sensor beneath the information about the trigger which sends an event;

After building the report you define the sensor boxes which are triggers and attach them on our box;

```
...
     NxQuat quat45(45.0f, NxVec3(0, 1, 0));

     NxMat33 m45;

     m45.id();

     m45.fromQuat(quat45);

     Sensor45Desc.name="45";

     Sensor45Desc.dimensions = NxVec3(2.5f, 0.1f, 0.1f);

     Sensor45Desc.localPose.t=NxVec3(1.5f, 0.0f, -1.5f);

     Sensor45Desc.localPose.M  = m45;

...
```

Figure 4.4 Fragment of Program Sensor Description

NxQuat is a data type for the engine which stands for quaternion and NxMat33 is a 3x3 Matrix which is build used for the rotation matrix. In Figure 4.4 dimensions of Sensor Description define the dimension of threshold values.

When we repeat that for every 45 degrees on front side of our robot box we have a box like this;

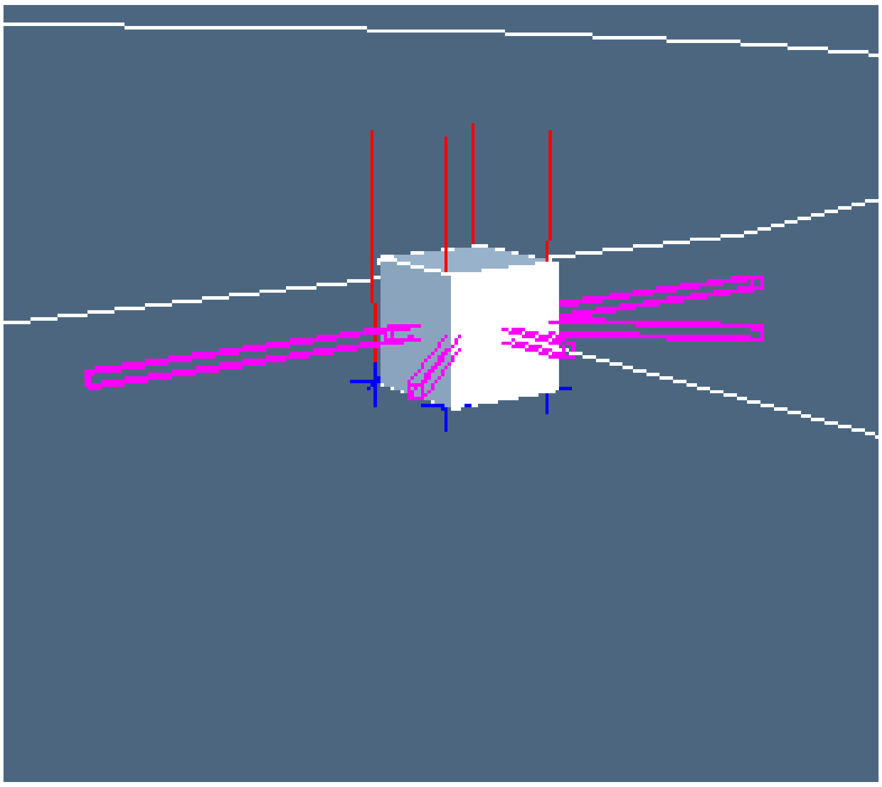


Figure 4.5 Robot with sensor signals.

In Figure 4.5 Magnet colors represent threshold distance of sensor signals and vertical edge flags let us visualize if cube hits an obstacle. Also Engine gives us the ability to visualize vector force which is applied by the moment over on it. We can visualize if the box hit some obstacle the edge flags fall down.

4.4 Robot Navigation

As it is expressed in flowchart of our program Figure 4.6 is a loop of decision making process. Robot first check if there is an obstacle around it and check if any sensor touched any obstacle, if it is false it enters the goal seeking behavior which is checked by if nbTouched > 0   then it tries to equalize the angle to destination and his orientation in local frame of it.

If robot find any obstacle it enters the obstacle avoidance behavior and first get the name of sensors if it touched and using rule table it avoids the obstacle. The flowchart representation of the program is shown in Figure 4.6

And the C++ code representation of the Obstacle Avoidance Table is;

```
if (sf.getD45s()||sf.getDm45s()||sf.getD0())
{    if(sf.getD0()){
       localForce.add(localForce,NxVec3(-200.0f,0.0f,0.0f));
       localSideForce.add
               (localSideForce,NxVec3(100.0f, 0.0f, 0.0f));
     }
     if (sf.getD45s()){

       localForce.add(localForce,NxVec3(-300.0f,0.0f,0.0f));
       localSideForce.add
               (localSideForce,NxVec3(200.0f, 0.0f, 0.0f)
     }
     if (sf.getDm45s()){
       localForce.add(localForce,NxVec3(-300.0f,0.0f,0.0f));
       localSideForce.add
              (localSideForce,NxVec3(-200.0f, 0.0f, 0.0f);
     }
  }
  else {
     if(sf.getD45()){
       localForce.add(localForce,NxVec3(100.0f, 0.0f, 0.0f));
       localSideForce.add
               (localSideForce,NxVec3(100.0f, 0.0f, 0.0f));
     }
     if (sf.getDm45()){
       localForce.add(localForce,NxVec3(100.0f, 0.0f, 0.0f));
       localSideForce.add
               (localSideForce,NxVec3(-100.0f, 0.0f, 0.0f));

     }
    if(sf.getD90()||sf.getDm90())
     {
       localForce.add(localForce,NxVec3(300.0f, 0.0f, 0.0f));
     }
   }
```

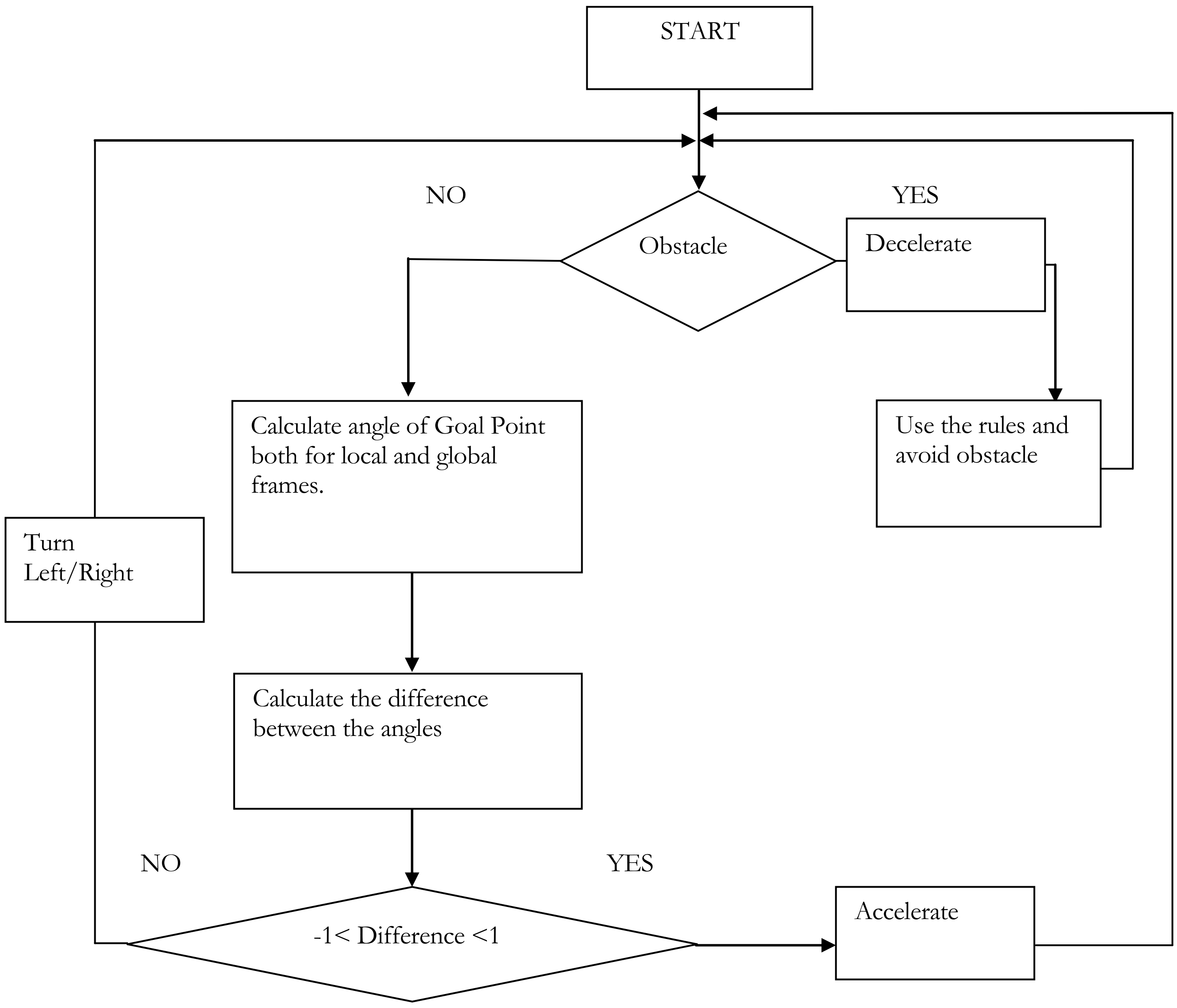
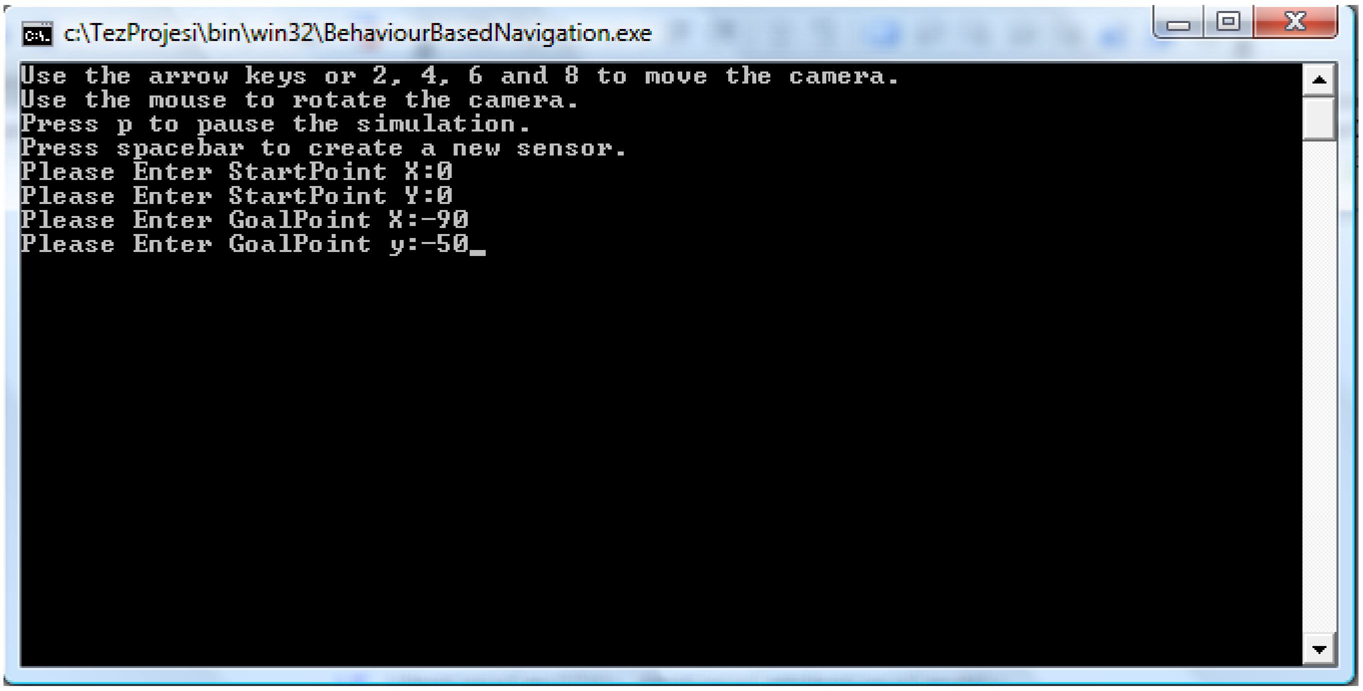

START
NO
YES
Obstacle
Decelerate
Calculate angle of Goal Point both for local and global frames.
Use the rules and avoid obstacle
Turn Left/Right
Calculate the difference between the angles
NO
YES
-1< Difference <1
Accelerate

Figure 4.6 Flowchart of Behavior Based Mobile Robot Navigation Program

Program starts by taking the goal point and start points from the user after and it enters the loop on every frame defined on the flowchart above. It first checks if there is an obstacle than if it finds an obstacle it enters the obstacle avoidance behavior and decelerate to V (safe) and starts to add local forces based on the gathered sensor information on robot until it avoids the obstacles. When the robot don't sense any obstacles around it, enters the goal seeking behavior and starts to turn the Goal Point when goal point angle on its local frame and global frame difference is lower than 1 or above than -1 it accelerate through the goal point. And it checks if there is an obstacle on every step.

The scenario as shown in Figures below uses the simple layout to test the obstacle avoidance and goal seeking behavior of the robot. The robot avoided the obstacles successfully. This suggested that the algorithm is suitable for obstacle avoidance. In simulator, obstacles are big cubes; therefore, the robot is regarded as a smaller cube with red sensor lines. There are five sensors for measuring distances. These distances are distance from Left Obstacle, distance from Right Obstacle, distance from Frontier Obstacles, distance from the obstacles on -45 degrees and distance from the obstacles on 45 degrees. These sensors help locating obstacles. The robot model in this simulator is an omnidirectional model robot which can go forward backward and stationary 360^o turning abilities and angular movement capabilities. We assume that the positions of obstacles are unknown and there are no pre-planned route points. In the program the obstacles are generated randomly as cube. At the beginning of program the start point and goal point coordinates are requested from user. Fig.4.7 demonstrates the result of entering start and goal position. After entering position 3D representation of scene (Fig.4.8) is appear on the screen. Goal is represented with Sphere Object. As a beginning manner robot turning to the goal and start moving. If robot encounters an obstacle try to avoid it using algorithm described in previous chapter. Fig.4.10 and Fig.4.11 describe obstacle avoidance scenes. Avoiding the obstacle robot come to the goal position. During avoiding obstacle and moving to goal robot calculate from position of robot and orientation of robot on real time (Fig.4.9).

Below Figures 4.12-4.16 show the movement sequences of robot first figure represent the start point for the robot than robot get in the goal seeking manner after equalizing the orientation angle and goal point angle on the robots local frame robot start to go through the goal. When obstacles are being detected cube turns to red and enter the obstacle avoidance behavior. After avoiding obstacles as a reason of goal seeking behavior our robot start to turn the goal point.



```
Use the arrow keys or 2, 4, 6 and 8 to move the camera.
Use the mouse to rotate the camera.
Press p to pause the simulation.
Press spacebar to create a new sensor.
Please Enter StartPoint X:0
Please Enter StartPoint Y:0
Please Enter GoalPoint X:-90
Please Enter GoalPoint y:-50_
```

Figure 4.7 Start of Program

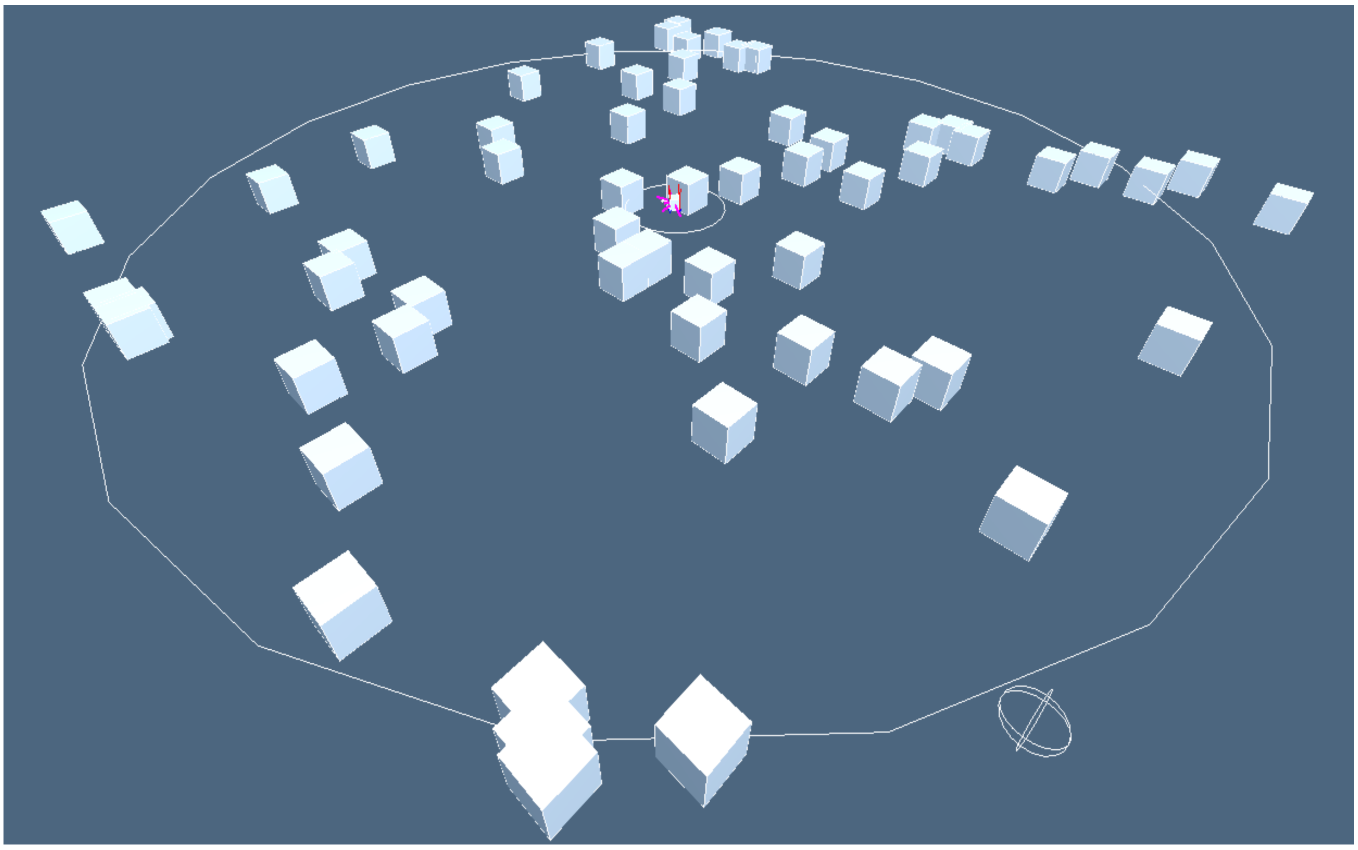
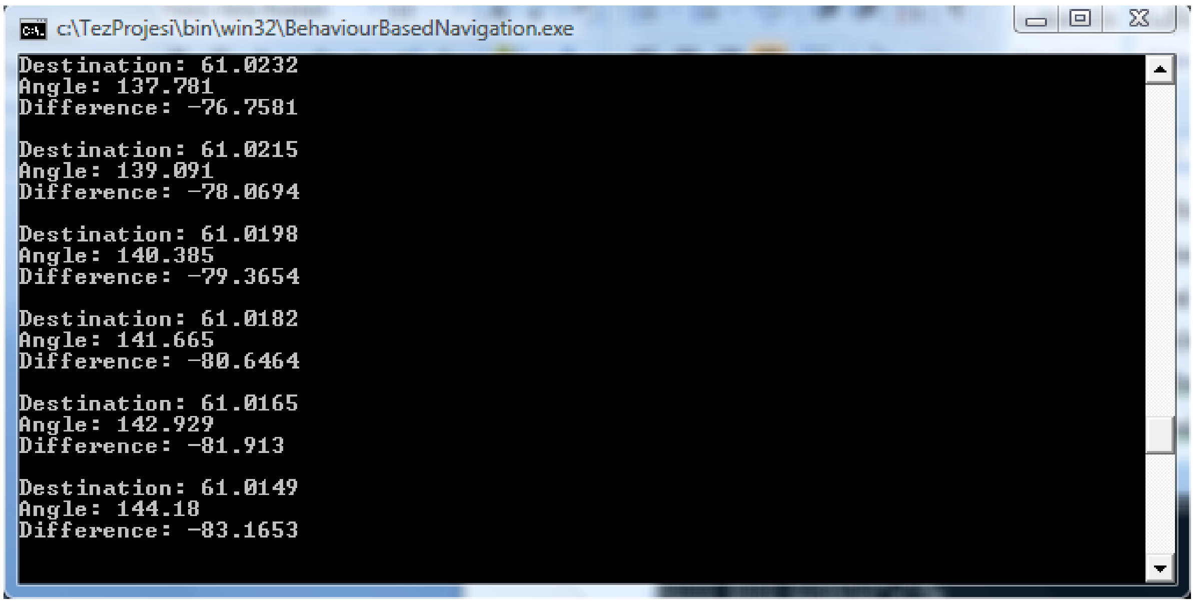


Figure 4.8 Robot at Start Point



```
Destination: 61.0232
Angle: 137.781
Difference: -76.7581

Destination: 61.0215
Angle: 139.091
Difference: -78.0694

Destination: 61.0198
Angle: 140.385
Difference: -79.3654

Destination: 61.0182
Angle: 141.665
Difference: -80.6464

Destination: 61.0165
Angle: 142.929
Difference: -81.913

Destination: 61.0149
Angle: 144.18
Difference: -83.1653
```

Figure 4.9 Real time calculations of Robot Orientation and Goal Point Angle

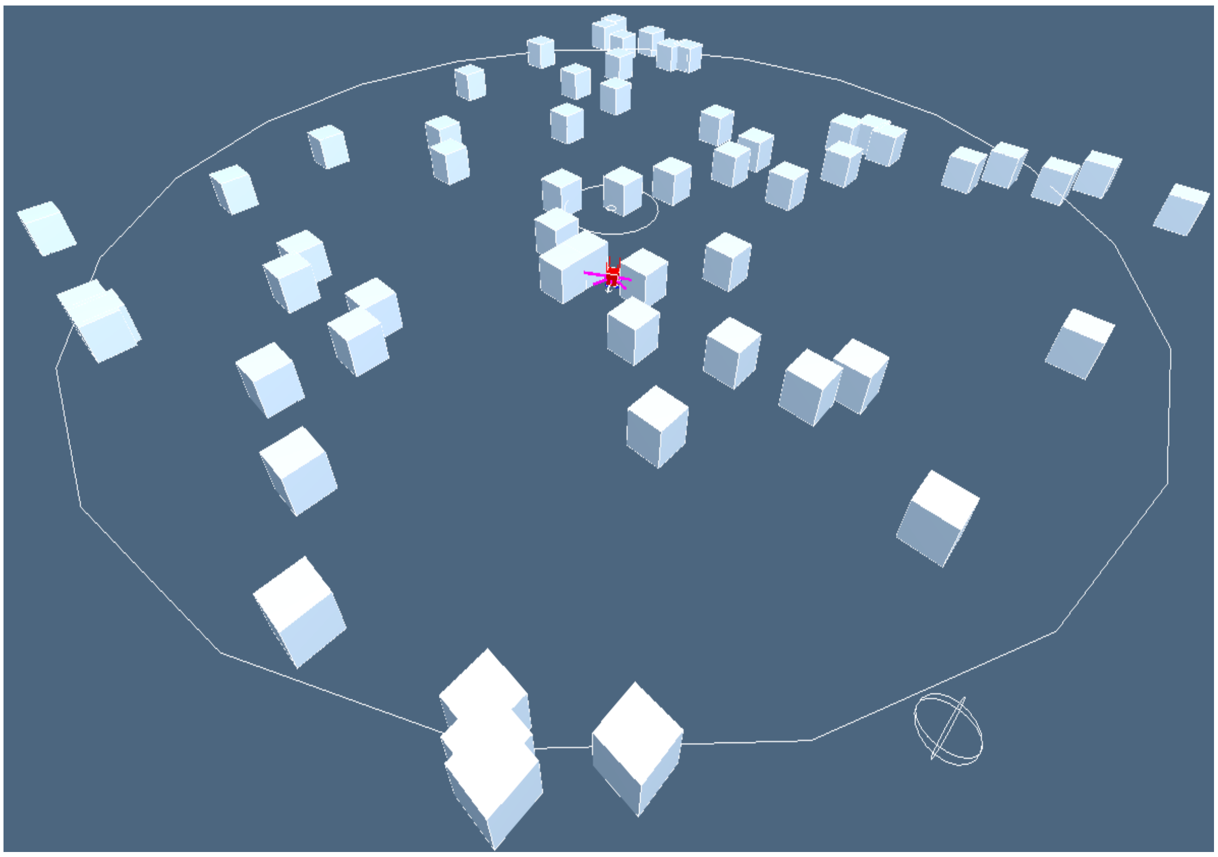


Figure 4.10 Robot Avoiding Obstacle

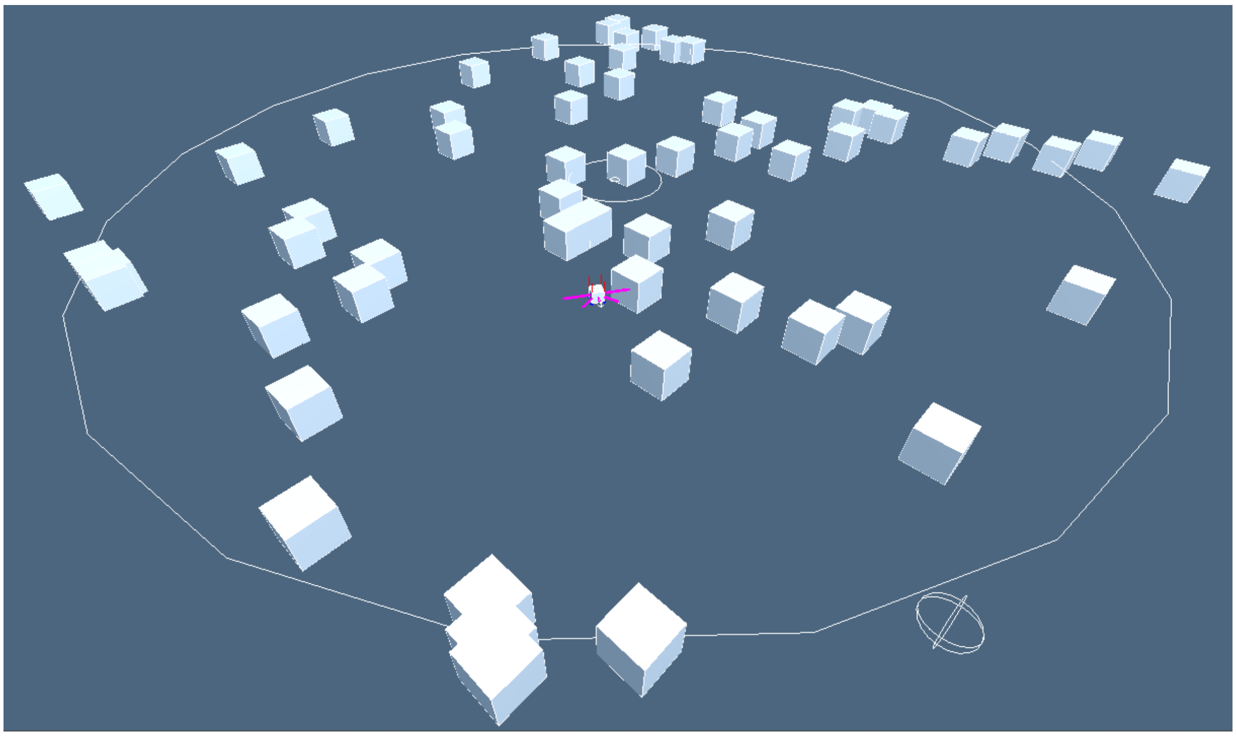


Figure 4.11 After Avoiding Obstacles Robot is on Goal-Seeking Behavior

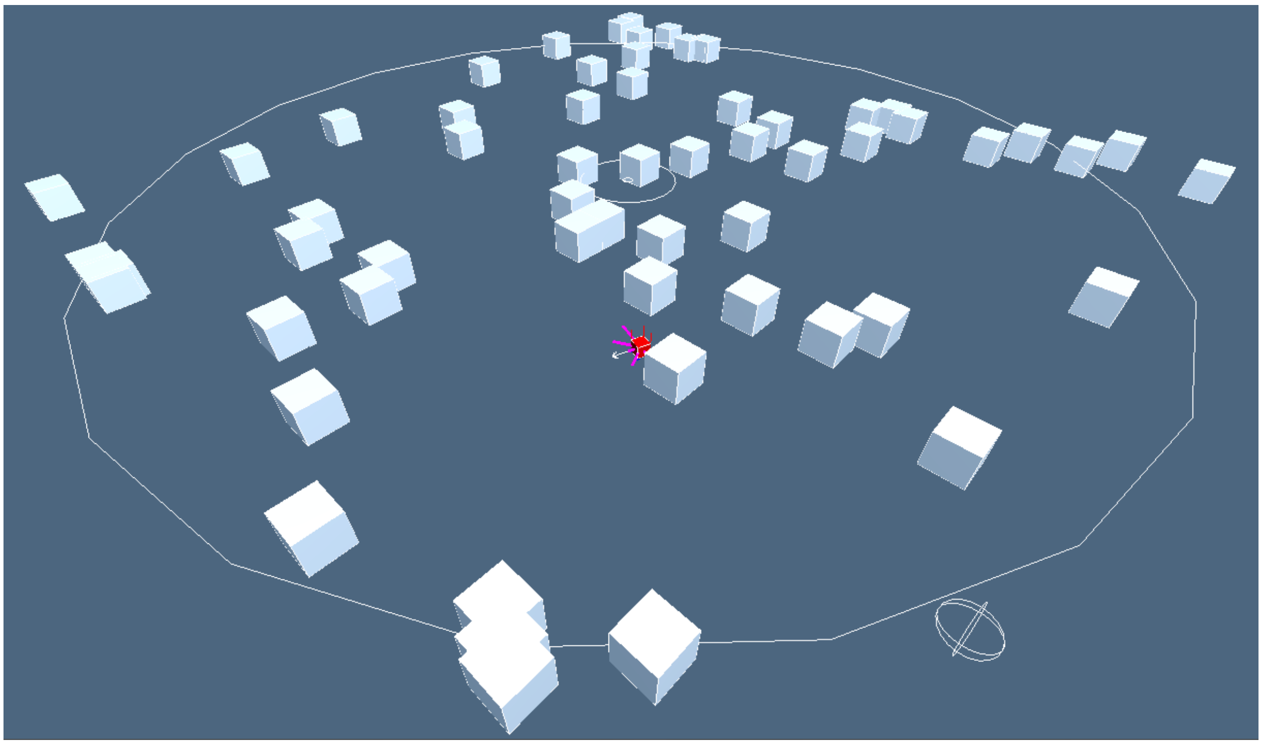


Figure 4.12 Robot Avoiding the Obstacle

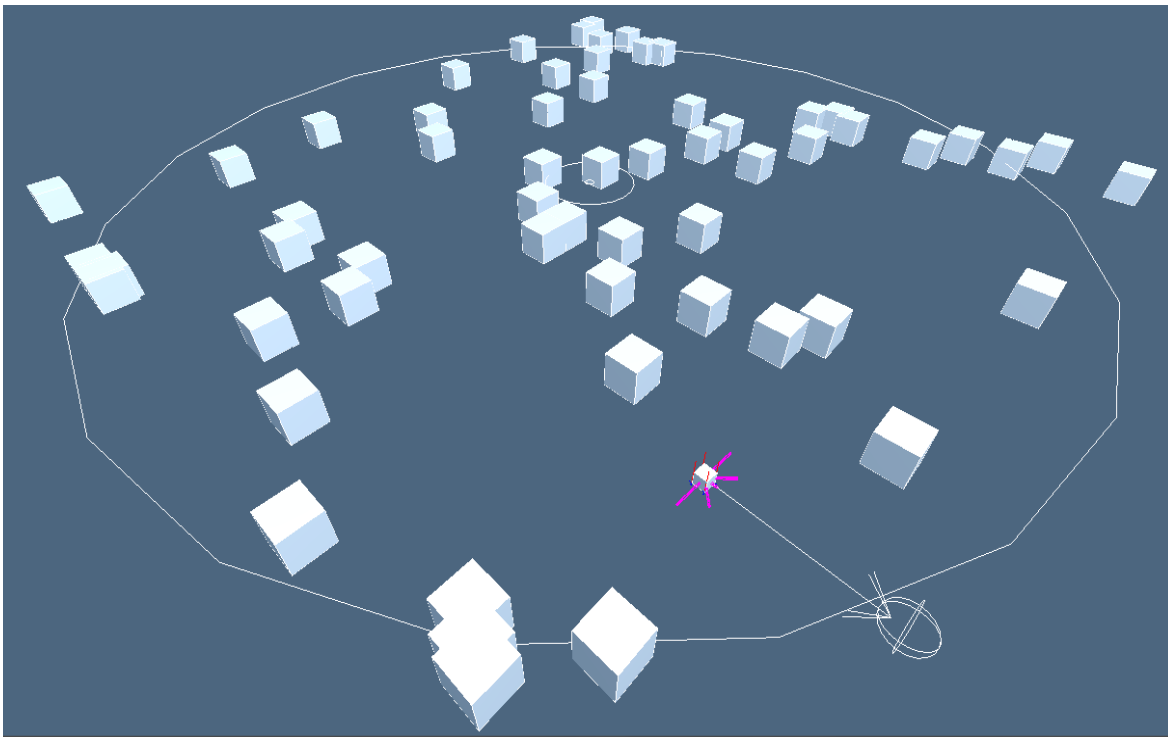


Figure 4.13 No Obstacle Detected and Robot Direct to Goal

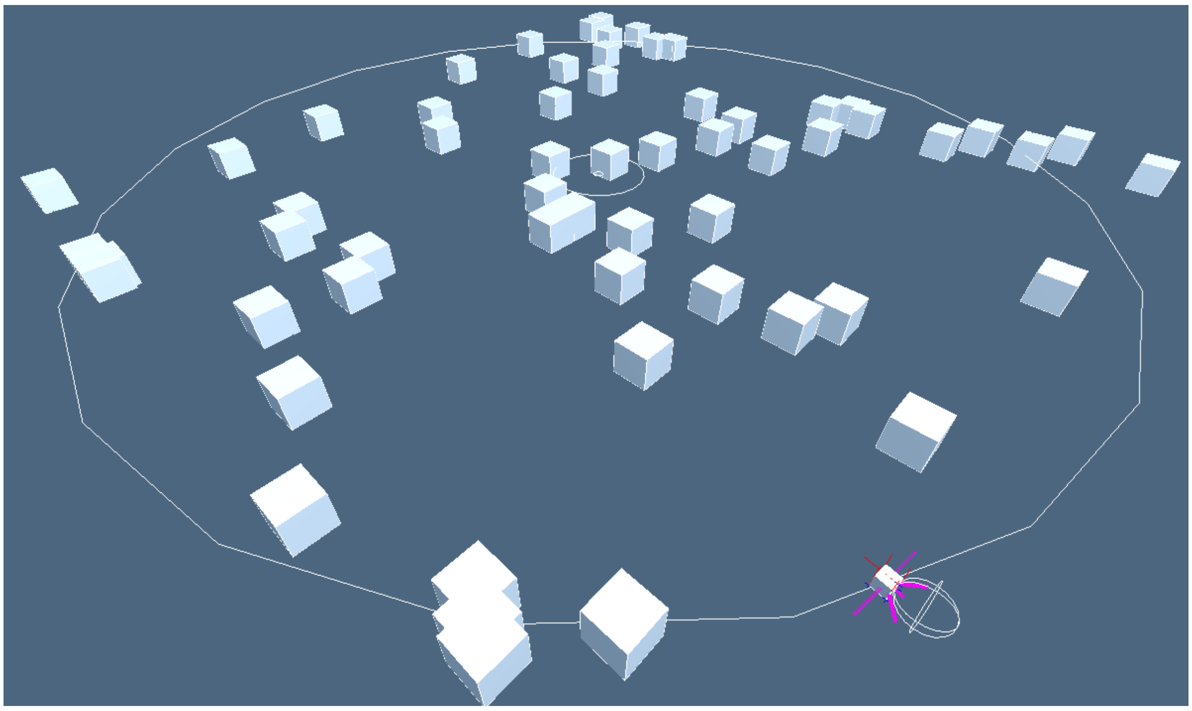


Figure 4.14 Goal Reached

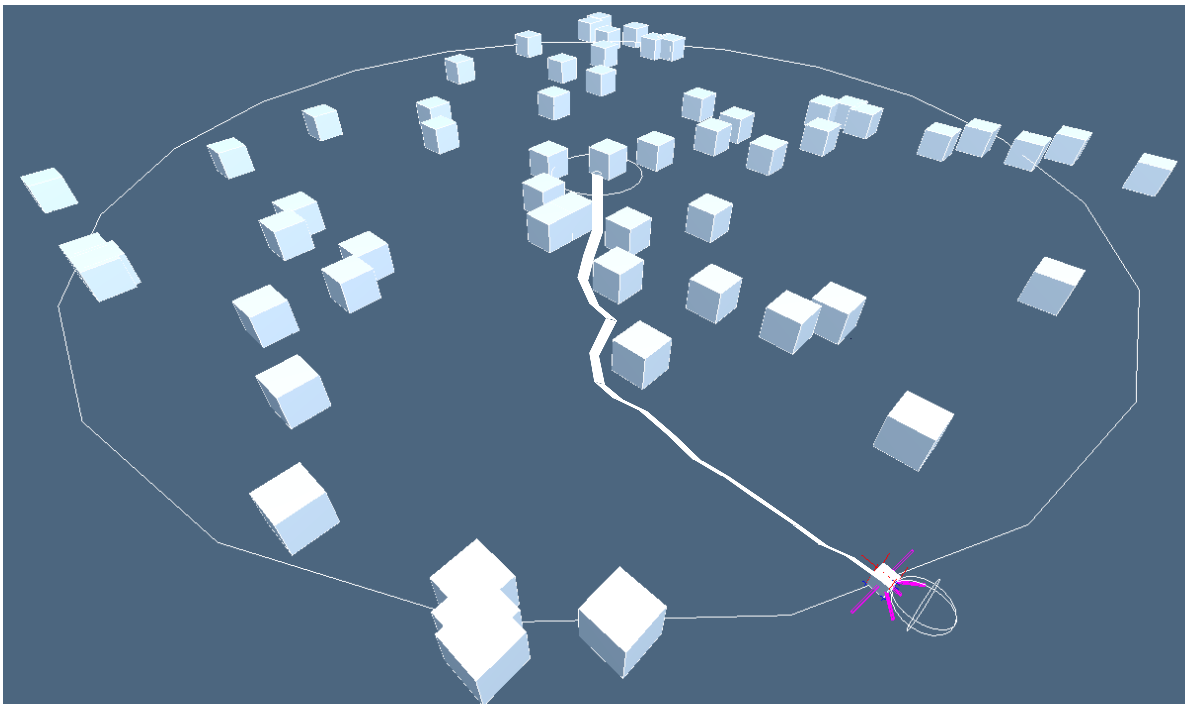


Figure 4.15 Path followed by the robot

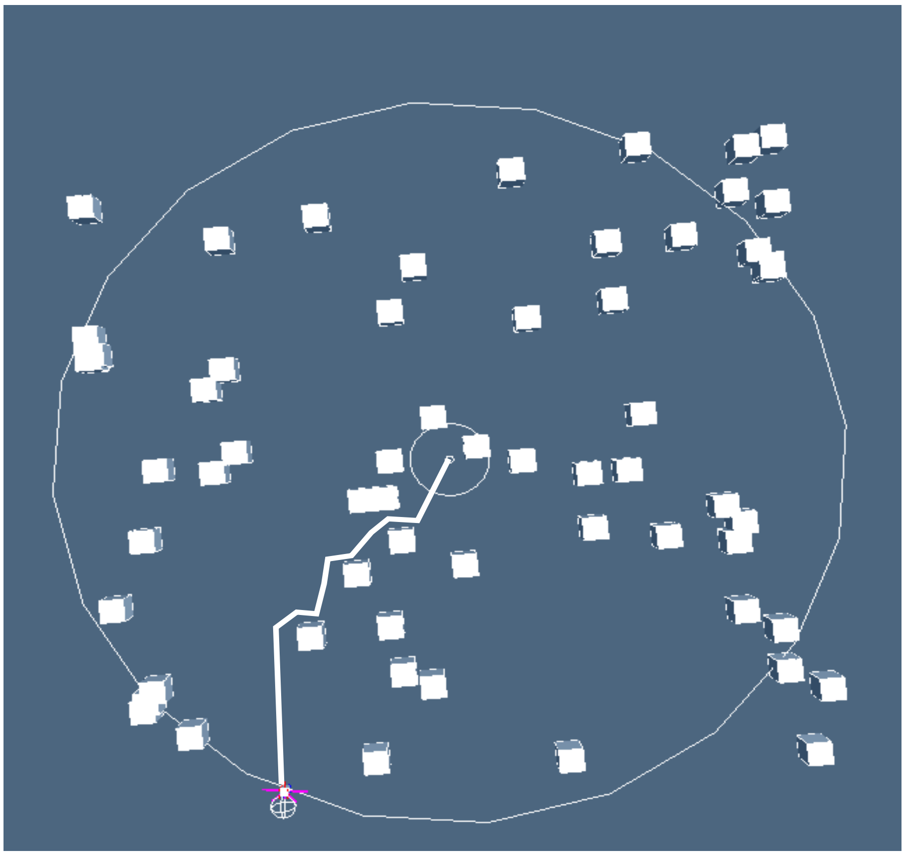


Figure 4.16 Above Scene

Experimental Results with U Shape Obstacle

Figure 4.17-4.20 demonstrate the avoiding of robot from U shaped, and L shaped obstacles.

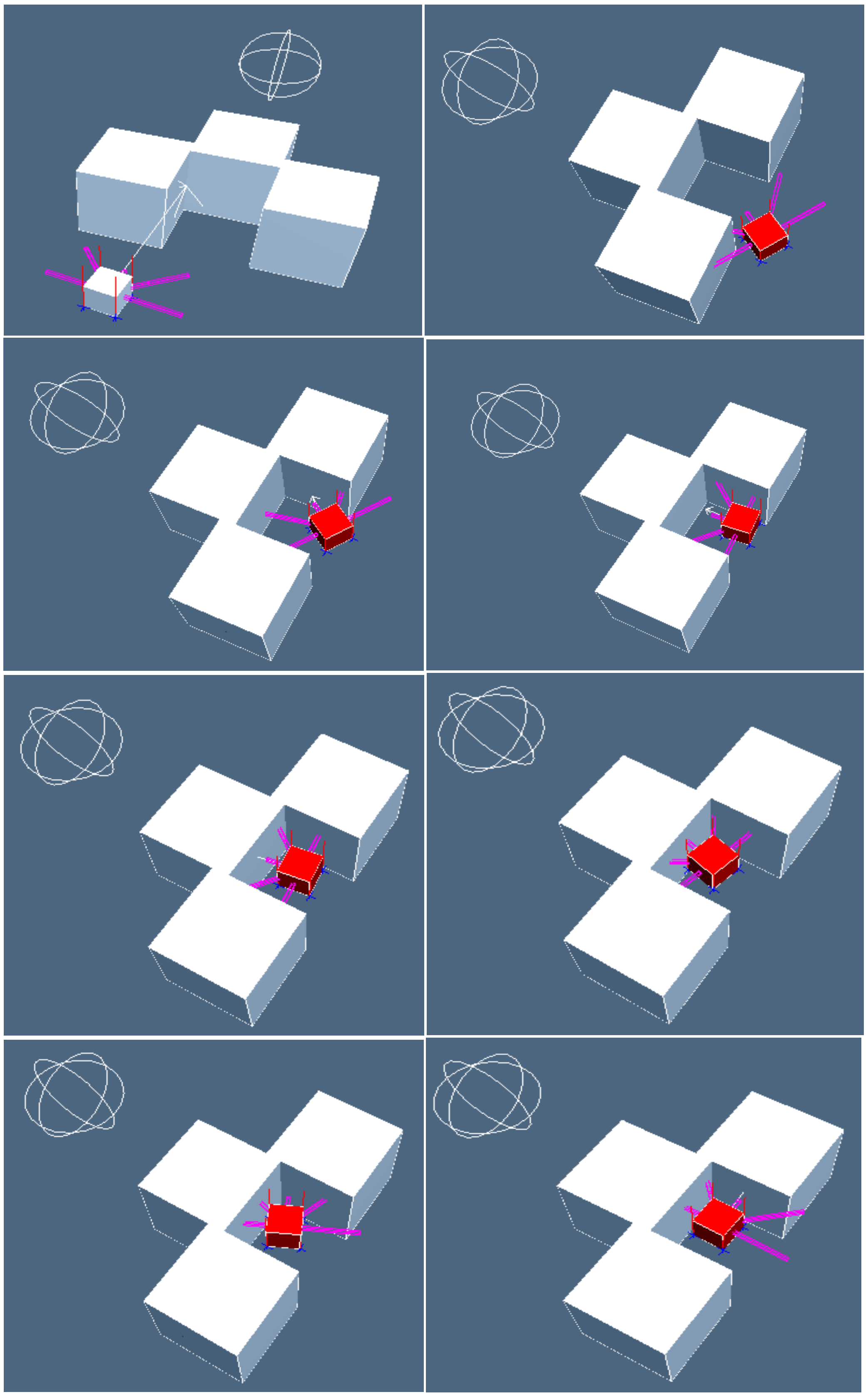


Figure 4.17 U Shape Object Obstacle Avoidance

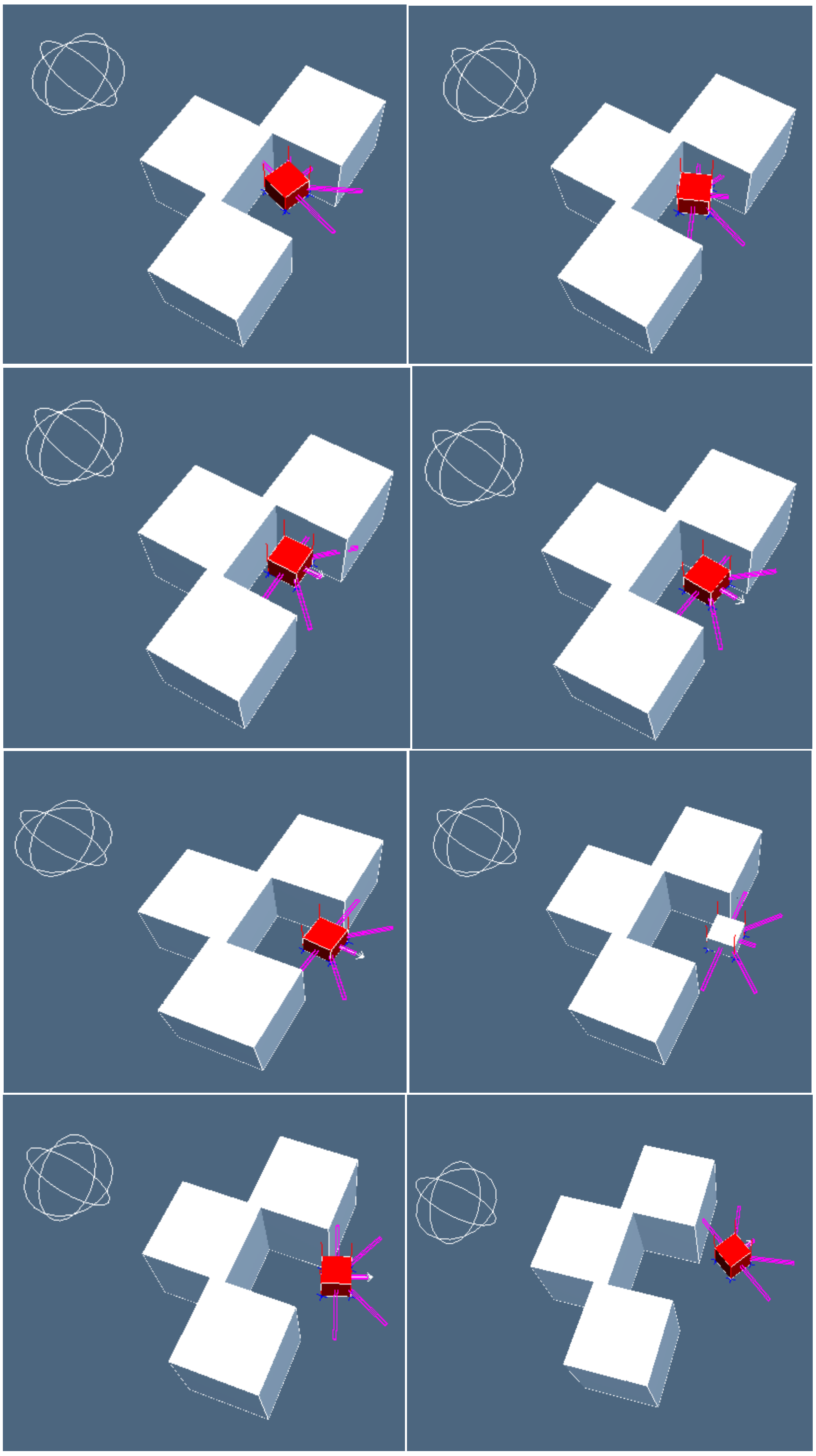


Figure 4.18 U Shape Object Obstacle Avoidance

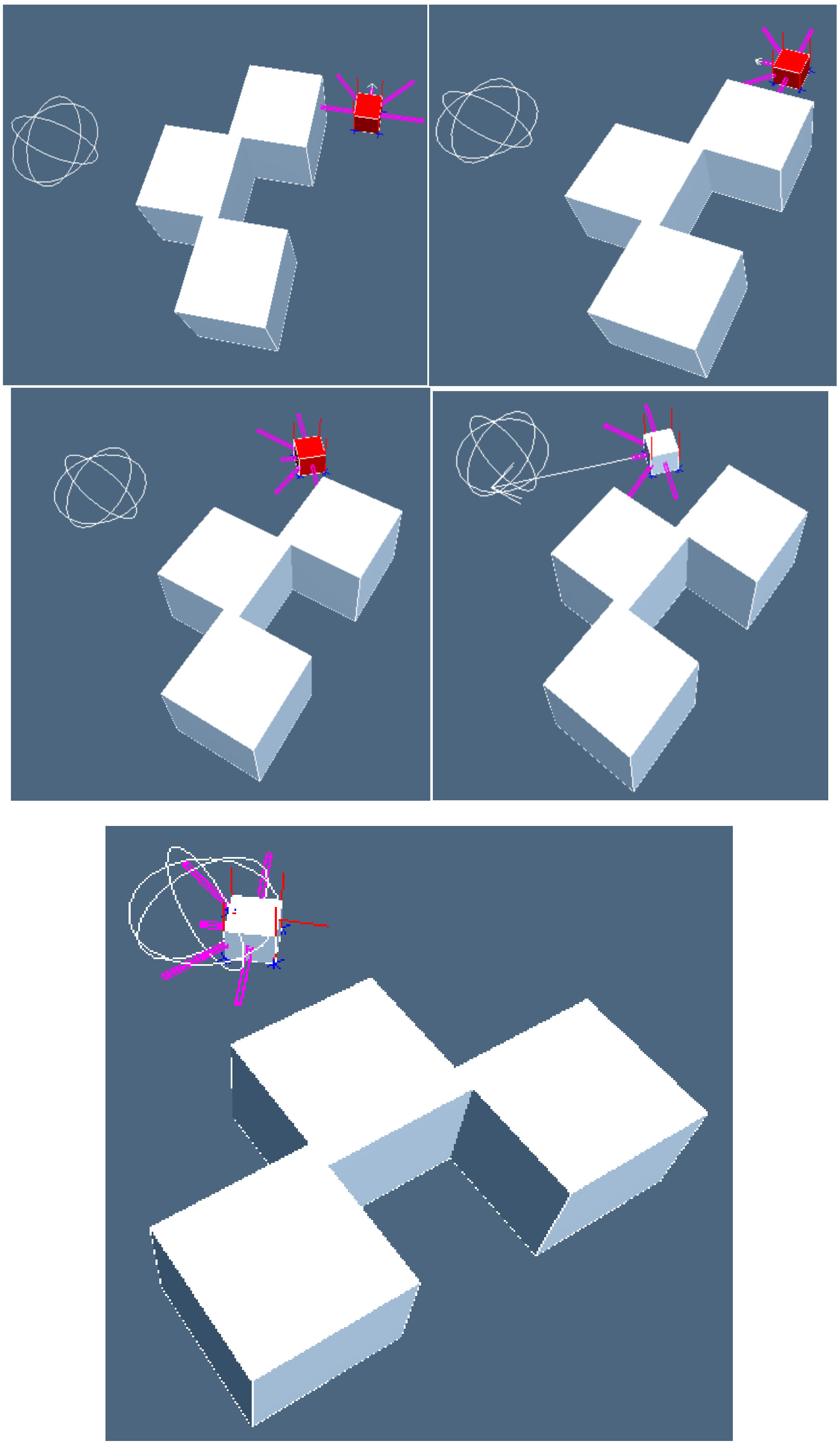
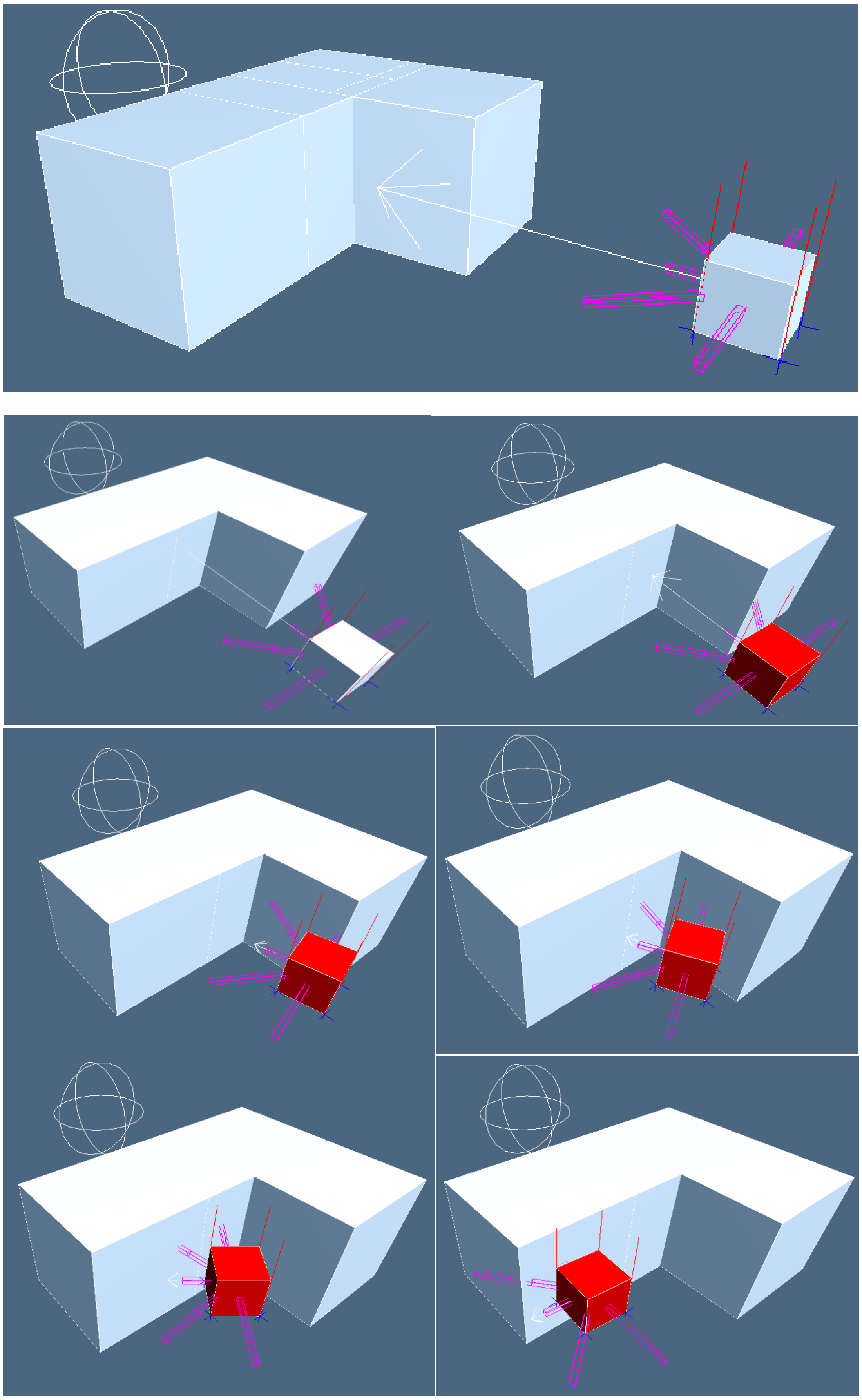


Figure 4.19 U Shape Object Obstacle Avoidance

Experimental Results on L Shape Obstacle



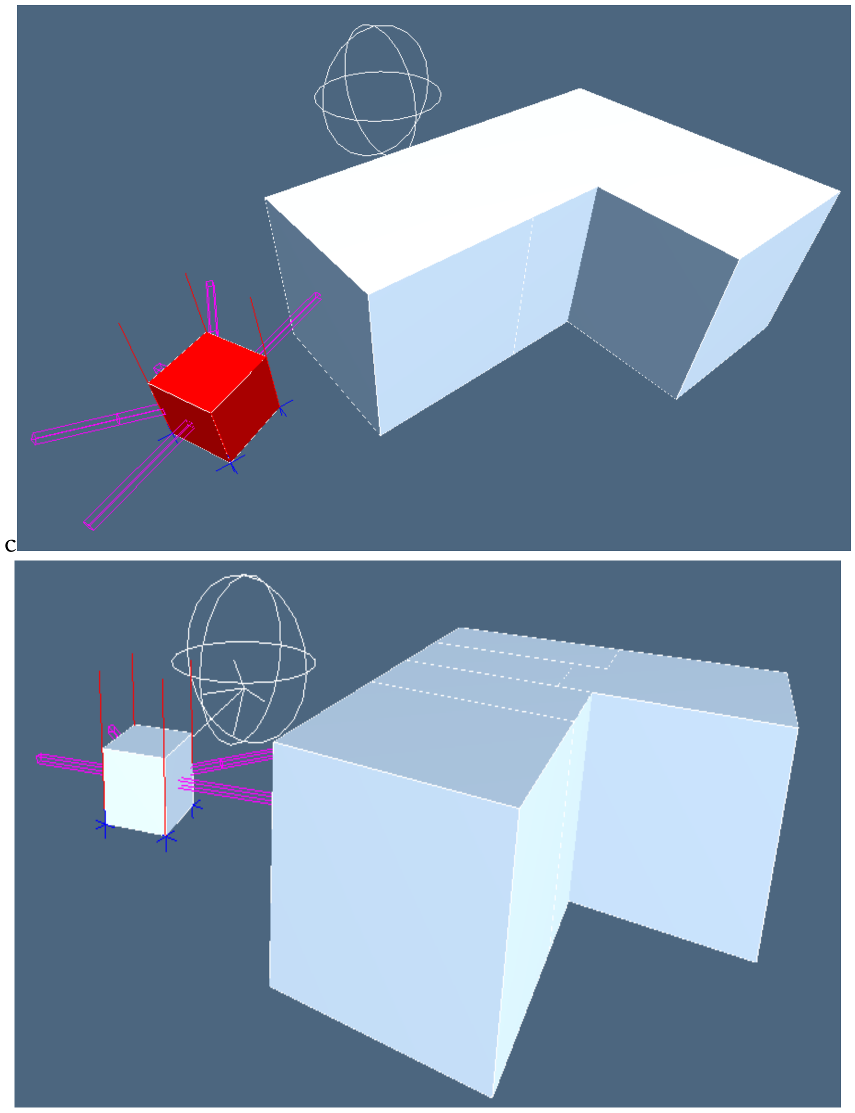


Figure 4.20 L Shape Object Obstacle Avoidance

4.5 Summary

The simulation of behavior based robot navigation has been done. The avoidance from obstacles and reaching goal are demonstrated. The Simulation has been done with Visual C++ Language using NVIDA PhysX SDK and Visual Studio 2005 IDE.

CONCLUSION

There are a number of robust techniques for various key sub-problems in robot navigation. There are also a wide variety 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.

The possibility of combining some of the more powerful techniques from each category, to result in a generic technique suitable to a wide variety of environments is still open.

In this study, we presented a new method for behavior fusion for robot navigation using behaviors. Since this method is to weight multiple reactive behaviors by rule tables.

Obstacle avoidance behavior and goal seeking behavior are the two most important behaviors that make up the functionality of mobile robot. Few scenarios with different arrangement of obstacles and goal used to test the capability of mobile robot.

Traditionally mobile robot projects have delayed handling moving objects in the environment beyond the scientific life of the project. The navigation algorithm has better reliability and real-time response since perception and decision units are integrated in one module and are directly oriented to a dynamic environment.

The simulation results show that the proposed method, only using information acquired by sensors, can perform robot navigation in complex and uncertain environments by weighting multiple reactive behaviors, such as avoiding obstacles, decelerating at curved and narrow roads, escaping from objects, and moving to target and so on. This method is suitable for robot navigation by multi-sensor integration.

The application of behavioral controller depends on the sensing strategies. The design and construction of the controller produces good theoretical results (the correct output) but this is fairly limited in scope without the ability to simulate it on the robot environment. But the benefit of it that provides a way to incrementally build and test a complex mobile robot control system. Besides leading to a different implementation strategy it is also interesting to note the way the decomposition affected the capabilities of the robot control system built in this study. In particular, our control system deals with moving objects in the environment at the very lowest level.

The results of this project clearly show that the behavior-based mobile robot is successfully developed using the rule based architecture. All the mobile robot behaviors coordinated well and interchanged smoothly with the proposed architecture. By using only five sensors, the main function of the mobile robot, obstacle avoidance, is giving a promising result.

REFERENCES

1. 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

2. Eric Abbott and David Powell.(1999,January) "*Land-Vehicle Navigation Using GPS*". Proceedings of the IEEE, Vol. 87, No. 1, pp. 145–162

3. Nourbakhsh, Roland Siegwart and Illah R. *Introduction to Autonomous Mobile Robots.* p.l. : The MIT Press Cambridge, Massachusetts.

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. Choset, Howie.(2005) Principles of Robot Motion: Theory, Algorithms, and Implementation. p.l. : The MIT Press .

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 Find 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. Ohno, K. Tsubouchi, T. Maeyama, S. Yuta (2003). *A mobile robot campus walkway following with daylight-change-proof walkway color image segmentation*. International Conference on Field and Service Robotics p.189-194.

18. B. Takeda, C Facchinetti, and J. C. Latombe (1992). *Planning the motions of a mobile robot in a sensory uncertainty field.* IEEE Transactions on Robotics and Automation.

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. 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.

26. 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.

27. Prahlad Vadakkepat, Kay Chen Tan and Wang Ming-Liang. *Evolutionary Artificial Potential Fields and Their Application in Real Time Robot Path.* p. 256.

28. 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.

29. 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.

30. 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.

31. Hamid Teimoori, Andrey V. Savkin (2008). *Equiangular navigation and guidance of a wheeled mobile robot based on range-only measurements.* 47th IEEE Conference on Decision and Control p.1747-1753.

32. 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.

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. Immanuel A. R. Ashokaraj, Peter M. G. Silson, Antonios Tsourdos, Brian A. White (2008, October).*Robust Sensor-Based Navigation for Mobile Robots.* IEEE Transactios on Instrumentation and Measurement,Vol. 58, No. 3, p. 551.

35.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.

36. Mahyuddin, Tan Tiong Cheng and Muhammad Nasiruddin (July 2009). *Implementation of Behaviour-Based Mobile Robot for Obstacle Avoidance Using a Single Ultrasonic Sensor.*Conference on Innovative Technologies in Intelligent Systems and Industrial Applications. p. 244 - 248. 25-26.

37. M. J. Mataric (1997). "*Behavior-based control: Examples from navigation, learning and group behavior*" Software Architecture Phys. Agents, vol. 9, p. 46–54.

38. Corporation, NVIDIA (2008). *PhysX Documentation.* Santa Clara, CA : NVIDIA.

39. 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.

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

41.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

42.X. Yang, M. Moallem, R.V. Patel (2005, August 28-31). *Motion Planning for Mobile Robots using a Fuzzy Layered Goal-Oriented Approach.* IEEE International Conference on Control Applications. p. 78-83.

43. J. Borenstein and Y. Koren (1990, 13-18 May) *Real-time obstacle avoidance for fast mobile robots in cluttered environments* IEEE International Conference on Robotics and Automation.

44. 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).

45. Jarvis., R. A (1994).*On distance transform based collision free path planning for robot navigation in known, unknown and time-varying environments.* Advanced Mobile Robots,.

46. D. Jung, G. Cheng, and A. Zelinsky (1997). *An Experiment in Realising Cooperative Behaviours between Autonomous Mobile Robots.* Barcelona, Spain : Fifth International Symposium On Experimental Robotics.

47. R. Abiyev, D. Ibrahim, and B. Erin (2009). EDURobot: *An Educational Computer Simulation Program for Navigation of Mobile Robots In The Presence Of Obstacles.* Near East University.

APPENDIX

Fragments of Program

```
#include <stdio.h>
#include <iostream>
#include <GL/glut.h>
#include <cstdlib>
#include <ctime>
// Physics code
#include "NxPhysics.h"
#include "ErrorStream.h"
#include "Timing.h"
#include "DebugRenderer.h"
#include "Utilities.h"
#include "SamplesVRDSettings.h"
#include "UserAllocator.h"
#include "3dInstance.h"


#ifdef __PPCGEKKO__
#include "GLFontRenderer.h"
#endif

static DebugRenderer  gDebugRenderer;
static bool         gPause = false;
static NxPhysicsSDK*  gPhysicsSDK = NULL;
static NxScene*       gScene = NULL;
static NxVec3       gDefaultGravity(0.0f, -98.1f, 0.0f);
static ErrorStream    gErrorStream;
static UserAllocator* gAllocator = NULL;
static NxVec3       RotationVec(0.0f,0.0f,0.0f);
// Render code
static int gMainHandle;
static bool gShadows = true;
static NxVec3 Eye(50.0f, 50.0f, 50.0f);
static NxVec3 Dir(-0.6,-0.2,-0.7);
static NxVec3 N;
static int mx = 0;
static int my = 0;
static oRifat::oMath Rmath;
NxVec3 StartPoint(0.0f, 0.0f, 0.0f);
NxVec3 GoalPoint(0.0f,0.0f,0.0f);
NxVec3 DirectionVec(0.0f,0.0f,1.0f);

char* Degree90="0";
char* Degree45="1";
char* Degree45s="2";
char* Degree0="3";
char* Degreem45="5";
char* Degreem45s="4";

char* Degreem90="6";
char* Control="7";

int Lx=0,Ly=0;
class MyObject
{
public: int    size;
     NxU32   nbTouched;
     void setForceTypeEnter(char*
ftype=Control){ForceTypeEnter=ftype;};
     void setForceTypeLeave(char*
ftype=Control){ForceTypeLeave=ftype;};
     char* getForceTypeEnter(){return ForceTypeEnter;};
     char* getForceTypeLeave(){return ForceTypeLeave;};
private: char* ForceTypeEnter;
      char* ForceTypeLeave;


};
oRifat::SensorFlags sf;


class SensorReport : public NxUserTriggerReport
{
  virtual void onTrigger(NxShape& triggerShape, NxShape& otherShape,
NxTriggerFlag status)
  {
    NX_ASSERT(triggerShape.getFlag(NX_TRIGGER_ENABLE));

    if(status & NX_TRIGGER_ON_ENTER)
    {
      NxActor& actor = otherShape.getActor();
      if(actor.userData)
      {
        NxActor& triggerActor = triggerShape.getActor();
        if(triggerActor.isDynamic() && triggerActor.userData !=
NULL)
        {
          MyObject* Object = (MyObject*)triggerActor.userData;
          Object->nbTouched++;
          Object-
>setForceTypeEnter((char*)triggerShape.getName());
          sf.Set((char*)triggerShape.getName());



        }
      }
    }
    if(status & NX_TRIGGER_ON_LEAVE)
    {
      NxActor& actor = otherShape.getActor();
      if(actor.userData)
      {
        NxActor& triggerActor = triggerShape.getActor();
        if(triggerActor.isDynamic() && triggerActor.userData !=
NULL)
        {
          MyObject* Object = (MyObject*)triggerActor.userData;
          Object-
>setForceTypeLeave((char*)triggerShape.getName());
          Object->nbTouched--;
          sf.Off((char*)triggerShape.getName());

        }
      }
    }
  }
}gMySensorReport;
```

```
static void CreateCube(const NxVec3& pos, int size=2, const NxVec3*
initial_velocity=NULL, bool isStatic=false)
{
  NxF32 a=10.0f;
  // Create body
  NxBodyDesc BodyDesc;
  BodyDesc.angularDamping = 1.5f;
  BodyDesc.angularVelocity.set(a);
  if(initial_velocity) BodyDesc.linearVelocity = *initial_velocity;

  NxBoxShapeDesc BoxDesc;
  BoxDesc.dimensions    = NxVec3(float(size), float(size),
float(size));

  MyObject* Object = new MyObject;
  Object->size       = size;
  Object->nbTouched     = 0;

  NxActorDesc ActorDesc;

  ActorDesc.userData    = Object;
  ActorDesc.shapes.pushBack(&BoxDesc);

  NxActorDesc Actor2Desc;
  Actor2Desc.userData     = Object;
  Actor2Desc.shapes.pushBack(&BoxDesc);

  NxBoxShapeDesc
Sensor0Desc,Sensor45Desc,Sensorm45Desc,Sensor45sDesc,Sensorm45sDesc,
Sensor90Desc,Sensorm90Desc;
  if(!isStatic)
  {
    BodyDesc.flags     |= NX_BF_FROZEN_ROT_X|NX_BF_FROZEN_ROT_Z;
    BodyDesc.linearDamping  = 0.5f;
    BodyDesc.angularDamping=3.0f;


    NxQuat quat0(0.0f, NxVec3(0, 1, 0));
    NxMat33 m0;
    m0.id();
    m0.fromQuat(quat0);
    Sensor0Desc.name="3";
    Sensor0Desc.dimensions  = NxVec3(0.65f, 0.1f, 0.1f);
    Sensor0Desc.localPose.t=NxVec3(1.5f, 0.0f, 0.0f);
    Sensor0Desc.localPose.M = m0;

    NxQuat quat45(45.0f, NxVec3(0, 1, 0));
    NxMat33 m45;
    m45.id();
    m45.fromQuat(quat45);
    Sensor45Desc.name="1";
    Sensor45Desc.dimensions = NxVec3(1.5f, 0.1f, 0.1f);
    Sensor45Desc.localPose.t=NxVec3(2.0f, 0.0f, -2.0f);
    Sensor45Desc.localPose.M  = m45;

    NxQuat quat90(90.0f, NxVec3(0, 1, 0));
    NxMat33 m90;
    m90.id();
    m90.fromQuat(quat90);
```

```
Sensor90Desc.name="0";
Sensor90Desc.dimensions = NxVec3(1.5f, 0.1f, 0.1f);
Sensor90Desc.localPose.t=NxVec3(0.0f, 0.0f, -2.5f);
Sensor90Desc.localPose.M  = m90;


NxQuat quatm90(-90.0f, NxVec3(0, 1, 0));
NxMat33 mm90;
mm90.id();
mm90.fromQuat(quatm90);
Sensorm90Desc.name="6";
Sensorm90Desc.dimensions  = NxVec3(2.0f, 0.1f, 0.1f);
Sensorm90Desc.localPose.t=NxVec3(0.0f, 0.0f, 2.5f);
Sensorm90Desc.localPose.M = mm90;


NxQuat quat45s(45.0f, NxVec3(0, 1, 0));
NxMat33 m45s;
m45s.id();
m45s.fromQuat(quat45s);
Sensor45sDesc.name="2";
Sensor45sDesc.dimensions  = NxVec3(1.5f, 0.1f, 0.1f);
Sensor45sDesc.localPose.t=NxVec3(0.65f, 0.0f, -0.65f);
Sensor45sDesc.localPose.M = m45s;

NxQuat quatm45s(-45.0f, NxVec3(0, 1, 0));
NxMat33 mm45s;
mm45s.id();
mm45s.fromQuat(quatm45s);
Sensorm45sDesc.name="4";
Sensorm45sDesc.dimensions = NxVec3(1.5f, 0.1f, 0.1f);
Sensorm45sDesc.localPose.t=NxVec3(0.65f, 0.0f, 0.65f);
Sensorm45sDesc.localPose.M= mm45s;

NxQuat quatm45(-45.0f, NxVec3(0, 1, 0));
NxMat33 mm45;
mm45.id();
mm45.fromQuat(quatm45);
Sensorm45Desc.name="5";
Sensorm45Desc.dimensions  = NxVec3(2.0f, 0.1f, 0.1f);
Sensorm45Desc.localPose.t=NxVec3(2.0f, 0.0f, 2.0f);
Sensorm45Desc.localPose.M = mm45;


Sensor0Desc.shapeFlags  |= NX_TRIGGER_ON_ENTER;
Sensor45Desc.shapeFlags |= NX_TRIGGER_ON_ENTER;
Sensorm45sDesc.shapeFlags |= NX_TRIGGER_ON_ENTER;
Sensor45sDesc.shapeFlags  |= NX_TRIGGER_ON_ENTER;
Sensorm45Desc.shapeFlags  |= NX_TRIGGER_ON_ENTER;
Sensor90Desc.shapeFlags |= NX_TRIGGER_ON_ENTER;
Sensorm90Desc.shapeFlags  |= NX_TRIGGER_ON_ENTER;




Sensor0Desc.shapeFlags  |= NX_TRIGGER_ON_LEAVE;
Sensor45Desc.shapeFlags |= NX_TRIGGER_ON_LEAVE;
Sensorm45Desc.shapeFlags  |= NX_TRIGGER_ON_LEAVE;
Sensorm45sDesc.shapeFlags |= NX_TRIGGER_ON_LEAVE;
Sensor45sDesc.shapeFlags  |= NX_TRIGGER_ON_LEAVE;
```

```
        Sensor90Desc.shapeFlags |= NX_TRIGGER_ON_LEAVE;
        Sensorm90Desc.shapeFlags  |= NX_TRIGGER_ON_LEAVE;



        ActorDesc.shapes.pushBack(&Sensor0Desc);
        ActorDesc.shapes.pushBack(&Sensor45Desc);
        ActorDesc.shapes.pushBack(&Sensorm45Desc);
        ActorDesc.shapes.pushBack(&Sensor45sDesc);
        ActorDesc.shapes.pushBack(&Sensorm45sDesc);
        ActorDesc.shapes.pushBack(&Sensor90Desc);
        ActorDesc.shapes.pushBack(&Sensorm90Desc);


    }

    if(!isStatic) ActorDesc.body= &BodyDesc;
    ActorDesc.density     = 10.0f;
    ActorDesc.globalPose.t  = pos;
    NxActor* newActor = gScene->createActor(ActorDesc);
}
static void CreateSphere(const NxVec3& pos, int size=2)
{
    NxSphereShapeDesc Sphr;
    Sphr.skinWidth=2.0f;
    Sphr.radius    = float(size);
    Sphr.density=float(size);
    NxActorDesc ActorDesc;


    ActorDesc.shapes.pushBack(&Sphr);
    ActorDesc.density     = 10.0f;

    ActorDesc.globalPose.t  = pos;
    NxActor* newActor = gScene->createActor(ActorDesc);
}
static bool InitNx()
{
    if (!gAllocator)
        gAllocator = new UserAllocator;

    // Initialize PhysicsSDK
    NxPhysicsSDKDesc desc;
    NxSDKCreateError errorCode = NXCE_NO_ERROR;
    gPhysicsSDK = NxCreatePhysicsSDK(NX_PHYSICS_SDK_VERSION,
gAllocator, &gErrorStream, desc, &errorCode);
    if(gPhysicsSDK == NULL)
    {
        printf("\nSDK create error (%d - %s).\nUnable to initialize the
PhysX SDK, exiting the sample.\n\n", errorCode,
getNxSDKCreateError(errorCode));
        return false;
    }
#if SAMPLES_USE_VRD
    // The settings for the VRD host and port are found in
SampleCommonCode/SamplesVRDSettings.h
    if (gPhysicsSDK->getFoundationSDK().getRemoteDebugger())
        gPhysicsSDK->getFoundationSDK().getRemoteDebugger()-
>connect(SAMPLES_VRD_HOST, SAMPLES_VRD_PORT, SAMPLES_VRD_EVENTMASK);
#endif
```

```
  gPhysicsSDK->setParameter(NX_SKIN_WIDTH, 0.025f);
  gPhysicsSDK->setParameter(NX_VISUALIZATION_SCALE, 4.0f);
  gPhysicsSDK->setParameter(NX_VISUALIZE_COLLISION_SHAPES, 1);

  // Visual Parameters
  gPhysicsSDK->setParameter(NX_VISUALIZE_CONTACT_POINT, 1);
  gPhysicsSDK->setParameter(NX_VISUALIZE_CONTACT_NORMAL, 1);
  gPhysicsSDK->setParameter(NX_VISUALIZE_CONTACT_FORCE, 1);
  gPhysicsSDK->setParameter(NX_VISUALIZE_BODY_LIN_VELOCITY, 1);
  gPhysicsSDK->setParameter(NX_VISUALIZE_BODY_ANG_FORCE, 1);
  gPhysicsSDK->setParameter(NX_VISUALIZE_ACTIVE_VERTICES,1);

  // Create a scene
  NxSceneDesc sceneDesc;
  sceneDesc.gravity         = gDefaultGravity;
  sceneDesc.userTriggerReport    = &gMySensorReport;
  gScene = gPhysicsSDK->createScene(sceneDesc);

  if(gScene == NULL)
  {
    printf("\nError: Unable to create a PhysX scene, exiting the
sample.\n\n");
    return false;
  }
  gScene->setTiming(1.0f/60.0f, 1, NX_TIMESTEP_FIXED);


  NxMaterial * defaultMaterial = gScene->getMaterialFromIndex(0);
  defaultMaterial->setRestitution(0.0f);
  defaultMaterial->setStaticFriction(0.0f);
  defaultMaterial->setDynamicFriction(0.0f);

  NxPlaneShapeDesc PlaneDesc;
  NxActorDesc ActorDesc;
  ActorDesc.shapes.pushBack(&PlaneDesc);
  gScene->createActor(ActorDesc);

  const int NB=30;
  const int SIZE=4;
  const float MAXSIZE=NB*SIZE*2.0f;
  unsigned int i;
  for(i=0;i<NB;i++)
  {
    CreateCube(NxVec3((NxF32(i)/NxF32(NB-1))*MAXSIZE -
MAXSIZE*0.5f, NxF32(SIZE), MAXSIZE*0.5f), SIZE, NULL, true);
  }

  for(i=0;i<NB;i++)
  {
    CreateCube(NxVec3((NxF32(i)/NxF32(NB-1))*MAXSIZE -
MAXSIZE*0.5f, NxF32(SIZE), -MAXSIZE*0.5f), SIZE, NULL, true);
  }
  for(i=0;i<NB;i++)
  {
    CreateCube(NxVec3(-MAXSIZE*0.5f, NxF32(SIZE),
(NxF32(i)/NxF32(NB-1))*MAXSIZE - MAXSIZE*0.5f), SIZE, NULL, true);
  }

  for(i=0;i<NB;i++)
  {
    CreateCube(NxVec3(MAXSIZE*0.5f, NxF32(SIZE),
(NxF32(i)/NxF32(NB-1))*MAXSIZE - MAXSIZE*0.5f), SIZE, NULL, true);
  }
  srand(TIME_SAMPLES);
  for(i=0;i<NB;i++)
  {

    CreateCube(NxVec3((float)(rand()%90),0,(float)(rand()%90)), 3,
NULL, true);
    CreateCube(NxVec3((float)(-1*rand()%90),0,(float)(rand()%90)),
3, NULL, true);
    CreateCube(NxVec3((float)(rand()%90),0,(float)(-1*rand()%90)),
3, NULL, true);
    CreateCube(NxVec3((float)(-1*rand()%90),0,(float)(-
1*rand()%90)), 3, NULL, true);

  }

  //U Shape Cube
  /*
  CreateCube(GoalPoint-NxVec3(0,0,10),3,0,true);
  CreateCube(GoalPoint-NxVec3(-6,0,15),3,0,true);
  CreateCube(GoalPoint-NxVec3(6,0,15),3,0,true);*/

  /* L Shape Cube
  CreateCube(GoalPoint-NxVec3(0,0,-3),3,0,true);
  CreateCube(GoalPoint-NxVec3(6,0,-3),3,0,true);
  CreateCube(GoalPoint-NxVec3(-6,0,-3),3,0,true);
  CreateCube(GoalPoint-NxVec3(-6,0,-8),3,0,true);*/


  /* Converse L Shape
  CreateCube(GoalPoint-NxVec3(0,0,-3),3,0,true);
  CreateCube(GoalPoint-NxVec3(-6,0,-3),3,0,true);
  CreateCube(GoalPoint-NxVec3(6,0,-3),3,0,true);
  CreateCube(GoalPoint-NxVec3(6,0,-8),3,0,true);*/


  /*
  CreateCube(GoalPoint-NxVec3(0,0,10),3,0,true);
  CreateCube(GoalPoint-NxVec3(-6,0,10),3,0,true);
  CreateCube(GoalPoint-NxVec3(3,0,10),3,0,true);
  CreateCube(GoalPoint-NxVec3(3,0,15),3,0,true);
  */

  CreateSphere(GoalPoint+NxVec3(0,2,0),3);
  CreateCube(StartPoint, 1.0f);
  return true;
}

static void ExitCallback()
{
  if(gPhysicsSDK != NULL)
  {
    if(gScene != NULL) gPhysicsSDK->releaseScene(*gScene);
    gScene = NULL;
    NxReleasePhysicsSDK(gPhysicsSDK);
    gPhysicsSDK = NULL;
  }

  if (gAllocator)
```

```
    {
        delete gAllocator;
        gAllocator = NULL;
    }
}

static void KeyboardCallback(unsigned char key, int x, int y)
{
    switch(key)
    {
        case 27:
            exit(0);
            break;
        case ' ':
            CreateCube(StartPoint, 1);
            break;
        case 'p':
        case 'P':
            gPause = !gPause;
            break;
        case 101: case '8':  Eye += Dir * 2.0f; break;
        case 103: case '2':  Eye -= Dir * 2.0f; break;
        case 100: case '4':  Eye -= N * 2.0f; break;
        case 102: case '6':  Eye += N * 2.0f; break;
    }
}

static void ArrowKeyCallback(int key, int x, int y)
{
    KeyboardCallback(key,x,y);
}


static void MouseCallback(int button, int state, int x, int y)
{
    mx = x;
    my = y;
}

static void MotionCallback(int x, int y)
{
    int dx = mx - x;
    int dy = my - y;

    Dir.normalize();
    N.cross(Dir,NxVec3(0,1,0));

    NxQuat qx(NxPiF32 * dx * 20/ 180.0f, NxVec3(0,1,0));
    qx.rotate(Dir);
    NxQuat qy(NxPiF32 * dy * 20/ 180.0f, N);
    qy.rotate(Dir);

    mx = x;
    my = y;
}

static void RenderCallback()
{
    if(!gScene) return;

    NxF32 elapsedTime = getElapsedTime();
```

```
   // Physics code
   if(!gPause)
     {
     if (elapsedTime != 0.0f)
       {
       gScene->simulate(elapsedTime);
       gScene->flushStream();
       gScene->fetchResults(NX_RIGID_BODY_FINISHED, true);
       }
     }

   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(60.0f,
(float)glutGet(GLUT_WINDOW_WIDTH)/(float)glutGet(GLUT_WINDOW_HEIGHT)
, 1.0f, 10000.0f);

   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();


   int nbActors = gScene->getNbActors();
   NxActor** actors = gScene->getActors();
   NxVec3 actorLastLocalPoint(0,0,0);
   NxVec3 actorLocalPoint =actors[nbActors-1]->getGlobalPosition();


   //Looking Stable
   //gluLookAt(Eye.x, Eye.y, Eye.z, Eye.x + Dir.x, Eye.y + Dir.y,
Eye.z + Dir.z, 0.0f, 1.0f, 0.0f);
   //Looking Last RobotBox
   gluLookAt(Eye.x, Eye.y, Eye.z,
actorLocalPoint.x,actorLocalPoint.y, actorLocalPoint.z, 0.0f, 1.0f,
0.0f);


   while(nbActors--)
   {
     NxActor* actor = *actors++;

     if(actor->userData != NULL)
     {


       MyObject* Object = (MyObject*)actor->userData;
       glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
       if(actor->isDynamic() && !gPause)
       {
         if(!Object-
>nbTouched||(!sf.getD0()&&!sf.getD45s()&&!sf.getDm45s()&&!sf.getD45(
)&&!sf.getDm45()&&!sf.getDm90()&&!sf.getD90()))
         {
           if
(!sf.getD0()&&!sf.getD45s()&&!sf.getDm45s()&&!sf.getD45()&&!sf.getDm
45()&&!sf.getDm90()&&!sf.getD90())
           {
             sf.Set(Degreem90);
           }
```

```
            //NxF32
destangle=Rmath.getAngle(Rmath.GPtoLP(actor,GoalPoint),NxVec3(0.0f,0
.0f,0.0f));
            NxF32 destangle=Rmath.getAngle(GoalPoint,actor-
     >getGlobalPosition());
            NxF32 angle=Rmath.getAngleFQ(actor);

            std::cout<<"Destination: "<<destangle;
            printf("\n");
            std::cout<<"Angle: "<<angle;
            printf("\n");
            std::cout<<"Difference: "<<destangle-angle;
            printf("\n");

            printf("\n");


            if (destangle<270) destangle=destangle+90;
            else destangle=destangle-270;


            if (angle<destangle-1.0f)
            {actor->addLocalForceAtLocalPos(NxVec3(-60.0f, 0.0f, -
60.0f), NxVec3(0.0f, 0.0f, -1.0f));
            }else if (angle>=destangle+1.0f)
            {actor->addLocalForceAtLocalPos(NxVec3(60.0f, 0.0f,
60.0f), NxVec3(0.0f, 0.0f, -1.0f));
            }
            else
            {
              actor->addLocalForceAtLocalPos(NxVec3(300.0f,0.0f,
0.0f), NxVec3(0.0f, 0.0f, 0.0f));//,NX_IMPULSE,false);
              actor->setLinearDamping(1.0f);

            }

          }
          else
          {
            NxVec3 localSidePos(-1.0f, 0.0f, -1.0f);
            NxVec3 localSideForce(0.0f, 0.0f, 0.0f);
            NxVec3 localPos(0.0f, 0.0f, 0.0f);
            NxVec3 localForce(0.0f, 0.0f, 0.0f);

          if (sf.getD45s()||sf.getDm45s()||sf.getD0())
          {
            if(sf.getD0())
            {
              localForce.add(localForce,NxVec3(-200.0f,0.0f,0.0f));
              localSideForce.add(localSideForce,NxVec3(100.0f, 0.0f,
0.0f));

            }
            if (sf.getD45s())
            {
              localSideForce.add(localSideForce,NxVec3(200.0f, 0.0f,
0.0f));

              localForce.add(localForce,NxVec3(-300.0f,0.0f,0.0f));

            }
            if (sf.getDm45s())
            {
              localSideForce.add(localSideForce,NxVec3(-200.0f,
0.0f, 0.0f));
              localForce.add(localForce,NxVec3(-300.0f,0.0f,0.0f));

            }

          }
          else
          {

            if(sf.getD45())
            {
              localSideForce.add(localSideForce,NxVec3(100.0f, 0.0f,
0.0f));

              localForce.add(localForce,NxVec3(100.0f, 0.0f, 0.0f));
            }
            if(sf.getD90()||sf.getDm90())
            {
              localForce.add(localForce,NxVec3(300.0f, 0.0f, 0.0f));
            }

            if (sf.getDm45())
            {
              localSideForce.add(localSideForce,NxVec3(-100.0f,
0.0f, 0.0f));
              localForce.add(localForce,NxVec3(100.0f, 0.0f, 0.0f));
            }

          }




            actor->addLocalForceAtLocalPos(localForce, localPos);
            actor-
>addLocalForceAtLocalPos(localSideForce,localSidePos);
            actor->setLinearDamping(5.0f);


            glColor4f(1.0f, 0.0f, 0.0f, 1.0f);
          }
        }

        glPushMatrix();
        float glmat[16];
        actor->getGlobalPose().getColumnMajor44(glmat);
        glMultMatrixf(glmat);

        glutSolidCube(float(Object->size)*2.0f);

        glPopMatrix();

        // Handle shadows
        if(gShadows)
        {
          glPushMatrix();

          const static float ShadowMat[]={ 1,0,0,0, 0,0,0,0, 0,0,1,0,
0,0,0,1 };
```

```
                glMultMatrixf(ShadowMat);
                glMultMatrixf(glmat);

                glDisable(GL_LIGHTING);
                glColor4f(0.1f, 0.2f, 0.3f, 1.0f);
                glutSolidCube(float(Object->size)*2.0f);
                glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
                glEnable(GL_LIGHTING);

                glPopMatrix();
            }
        }
    }
    gDebugRenderer.renderData(*gScene->getDebugRenderable());

#ifdef __PPCGEKKO__
    char buf[256];
      sprintf(buf,
      "Use the arrow keys to move the camera.\n"
    "Press a to pause the simulation.\n"
    "Press b to create a new sensor.\n");

      GLFontRenderer::setScreenResolution(glutGet(GLUT_WINDOW_WIDTH),
glutGet(GLUT_WINDOW_HEIGHT));
    GLFontRenderer::setColor(0.9f, 1.0f, 0.0f, 1.0f);
    GLFontRenderer::print(0.01, 0.9, 0.036, buf, false, 11, true);
#endif

    glutSwapBuffers();
}

static void ReshapeCallback(int width, int height)
{
    glViewport(0, 0, width, height);
}

static void IdleCallback()
{
    glutPostRedisplay();
}

int main(int argc, char** argv)
{
#ifdef __PPCGEKKO__
    printf("Use the arrow keys to move the camera.\n");
    printf("Press a to pause the simulation.\n");
    printf("Press b to create a new sensor.\n");
#else
    printf("Use the arrow keys or 2, 4, 6 and 8 to move the
camera.\n");
    printf("Use the mouse to rotate the camera.\n");
    printf("Press p to pause the simulation.\n");
    printf("Press spacebar to create a new sensor.\n");
    printf("Please Enter StartPoint X:");
    std::cin>>StartPoint.x;
    printf("Please Enter StartPoint Y:");
    std::cin>>StartPoint.z;
    printf("Please Enter GoalPoint X:");
    std::cin>>GoalPoint.x;
    printf("Please Enter GoalPoint Y:");
    std::cin>>GoalPoint.z;
```

```
#endif

   // Initialize Glut
   glutInit(&argc, argv);

   glutInitWindowSize(800, 600);
   glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
   gMainHandle = glutCreateWindow("Behaviour-Based Obstacle Avoidance
And Navigation");
   glutSetWindow(gMainHandle);
   //glutVideoResize(100,100,1024,768);
   //glutFullScreen();
   glutDisplayFunc(RenderCallback);
   glutReshapeFunc(ReshapeCallback);
   glutIdleFunc(IdleCallback);
   glutKeyboardFunc(KeyboardCallback);
   glutSpecialFunc(ArrowKeyCallback);
   glutMouseFunc(MouseCallback);
   glutMotionFunc(MotionCallback);
   MotionCallback(0,0);
   atexit(ExitCallback);
#ifdef __PPCGEKKO__
   glutRemapButtonExt('b', ' ');
   glutRemapButtonExt('a', 'p');
   glutRemapButtonExt(GLUT_KEY_UP, '8');
   glutRemapButtonExt(GLUT_KEY_DOWN, '2');
   glutRemapButtonExt(GLUT_KEY_LEFT, '4');
   glutRemapButtonExt(GLUT_KEY_RIGHT, '6');
   glutRemapButtonExt('1', '7');
   glutRemapButtonExt('2', '3');
#endif

   // Setup default render states
   glClearColor(0.3f, 0.4f, 0.5f, 1.0);
   glEnable(GL_DEPTH_TEST);
   glEnable(GL_COLOR_MATERIAL);
#ifdef __PPCGEKKO__
   glEnable(GL_CULL_FACE);
#endif
   // Setup lighting
   glEnable(GL_LIGHTING);
   float AmbientColor[]  = { 0.0f, 0.1f, 0.2f, 0.0f };
   glLightfv(GL_LIGHT0, GL_AMBIENT, AmbientColor);
   float DiffuseColor[]  = { 1.0f, 1.0f, 1.0f, 0.0f };
   glLightfv(GL_LIGHT0, GL_DIFFUSE, DiffuseColor);
   float SpecularColor[] = { 0.0f, 0.0f, 0.0f, 0.0f };
   glLightfv(GL_LIGHT0, GL_SPECULAR, SpecularColor);
   float Position[]    = { 100.0f, 100.0f, 400.0f, 1.0f };
   glLightfv(GL_LIGHT0, GL_POSITION, Position);
   glEnable(GL_LIGHT0);

   // Initialize physics scene and start the application main loop if
scene was created
   if (InitNx())
     glutMainLoop();

   return 0;
}
```

Fragment of “3dInstance.h” Header File

```
#include <stdio.h>
#include <iostream>
#include <GL/glut.h>

// Physics code
#include "NxPhysics.h"
#include "ErrorStream.h"
#include "Timing.h"
#include "DebugRenderer.h"
#include "Utilities.h"
#include "SamplesVRDSettings.h"
#include "UserAllocator.h"

namespace oRifat
{
  class oMath
  {
  private:NxMath math;
  public:NxF32 getAngle(NxVec3 FirstVec,NxVec3 SecondVec)
        {
          NxF32 angle;
          angle=math.radToDeg(math.atan2(SecondVec.x-
FirstVec.x,SecondVec.z-FirstVec.z));
          if (angle>0)return angle;
          else return (angle+360);

        }

        NxF32 CalculateAngel(NxActor &actor, NxReal &Angle)

        {

          NxVec3 Axis(0.0f,1.0f,0.0f);
          actor.getGlobalOrientationQuat().getAngleAxis(Angle,Axis);
          return Angle;



        }

        NxF32 getAngleFQ(NxActor* actor)
        {
          NxF32 angle;

          angle= math.radToDeg(actor-
>getGlobalOrientationQuat().getAngle());
          return angle;


        }
        NxF32 getAngleFQ(NxQuat q)
        {
          NxF32 angle;
          angle=math.radToDeg(q.getAngle());
          if (angle>0) return angle;
          else return (-1*angle);
        }

        NxF32 getAngle(NxVec3 SecondVec)
        {
          NxF32 angle;
          angle=math.radToDeg(math.atan2(0-SecondVec.x,0-
SecondVec.z));
          return angle;


        }

        NxF32 getDistance(NxVec3 FirstVec,NxVec3 SecondVec)
        {

          return math.sqrt(math.pow(SecondVec.z-
FirstVec.z,2)+math.pow(SecondVec.z-FirstVec.z,2));


        }

        NxVec3 ActorsLPToGP(NxActor* actor,NxVec3 LocalPoint)/*Turns
an actor's Local Point to a Global Point*/

        {
          NxMat33 transformationMatrix =actor-
>getGlobalOrientation();
          //transformationMatrix.setTransposed();


          return (transformationMatrix%LocalPoint);
        }

        NxF32 getFrontAngle(NxActor* actor,NxVec3 GoalPoint)
        {
          return getAngle(ActorsLPToGP(actor,
                    GoalPoint-actor->getGlobalPosition()),GoalPoint);
        }
   };
char* Degree90="0";
char* Degree45="1";
char* Degree45s="2";
char* Degree0="3";
char* Degreem45="5";
char* Degreem45s="4";

char* Degreem90="6";
char* Control="7";


   class SensorFlags
   {
   private:
   bool D45;
   bool Dm45;
   bool D45s;
   bool Dm45s;

   bool D0;
   bool D90;
   bool Dm90;
```

```
public:
  void offm45()
  {
    Dm45=0;
  }
  void off45()
  {
    D45=0;
  }
  void offm45s()
  {
    Dm45s=0;
  }
  void off45s()
  {
    D45s=0;
  }

  void off90()
  {
    D90=0;
  }
  void offm90()
  {
    Dm90=0;
  }
  void off0()
  {
    D0=0;
  }

  void SetAllSensorsDown()
  {
    D45=0;
    Dm45=0;
    D45s=0;
    Dm45s=0;
    D0=0;
    D90=0;
    Dm90=0;
    printf("\n All sensor flags down.\n");

  }
  bool getD0()
  {
    return D0;
  }
  bool getD45()
  {
    return D45;
  }
  bool getDm45()
  {
    return Dm45;
  }
  bool getD45s()
  {
    return D45s;
  }
  bool getDm45s()
  {
```

```
  return Dm45s;
}

bool getD90()
{
  return D90;
}
bool getDm90()
{
  return Dm90;
}
void Set(char* F)
{
      if(*F==*Degree45)
      {
          printf("45 ");
          D45=1;
      }
      if(*F==*Degree45s)
      {
        printf("45s ");
        D45s=1;
      }

      if (*F==*Degreem45s)
      {
        printf("-45s ");
        Dm45s=1;
      }
      if (*F==*Degreem45)
      {
        printf("-45 ");
        Dm45=1;
      }
      if(*F==*Degree0)
      {
        printf("0 ");
        D0=1;
      }
      if(*F==*Degree90)
      {
        printf("90 ");
        D90=1;
      }
      if(*F==*Degreem90)
      {
        printf("-90 ");
        Dm90=1;
      }
}

  void Off(char* F)
{


      if(*F==*Degree45)
      { off45();
        printf("45Off ");
      }
      if(*F==*Degree45s)
      { off45s();
```

```
                  printf("45sOff ");
                }

                 if  (*F==*Degreem45s)
                {
                  offm45s();
                  printf("-45sOff ");
                }
                  if  (*F==*Degreem45)
                {
                  offm45();
                  printf("-45Off ");
                }
                if(*F==*Degree0)
                {
                  off0();
                  printf("0Off ");
                }
                if(*F==*Degree90)
                {
                  off90();
                  printf("90Off ");
                }
                if(*F==*Degreem90)
                {
                  offm90();
                  printf("-90Off ");
                }
         }

    };
  }
```