uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

u Ottawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Martin Soucy
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Manipulator path planning with multi-resolution potential fields and fuzzy logic control

TITRE DE LA THÈSE / TITLE OF THESIS

P. Payeur
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Waiel Gueaieb

Peter Liu

Gary W. Slater
LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

MANIPULATOR PATH PLANNING WITH MULTI-RESOLUTION POTENTIAL FIELDS AND FUZZY LOGIC CONTROL

By Martin Soucy

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.A.Sc. degree in Electrical Engineering

School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa

Your file Votre référence
ISBN: 0-494-11414-2
Our file Notre référence
ISBN: 0-494-11414-2

TABLE OF CONTENT

Table of figures

Table of tables

ABSTRACT

Robot path planning is an important part of the development of autonomous systems. Numerous strategies have been proposed in the literature regarding mobile robots but trajectory planning for manipulators is considerably more difficult since the entire structure can move and therefore produce collisions with surrounding obstacles.

This thesis presents an original solution and analytical comparison to path planning for manipulator arms. Path planning is executed in two parts: first, a global path is found to guide the end effector in the environment using artificial potential fields and multi-resolution occupancy grids, then, a local path is determined for the entire robot structure by considering the kinematics of the robot as well as the repulsive forces of nearby obstacles in a fuzzy logic controller. Results are shown from a simulator that has been built for this purpose.

The contribution of this research is to develop a robust solution for path planning with collision avoidance: one that can be used for various manipulator arms and environment configurations. The proposed method is easily applicable to 2D and 3D environments and has been tested on both.

ACKNOWLEDGEMENTS

None of this work would have been possible without the help of numerous people. First and foremost, I would like to thank my research supervisor Dr. Pierre Payeur. His patience, guidance and understanding allowed me to carry out this research.

Also, the support given by my family and friends proved invaluable. Their sustained encouragement provided me the strength to complete this thesis.

I owe a great deal of gratitude to these people.

CHAPTER 1: INTRODUCTION

Robotic systems are becoming an integral part of our society. In manufacturing, robots have completely changed the way goods are made. But this is not the end of robotic technology. For many reasons, such as human safety or economics, it is desirable to have robots able to complete complex tasks with minimal or no human intervention. However, this is still far from being accomplished.

Moving a robot is a complex operation that requires a lot of knowledge about the environment. This information must be analyzed and interpreted to conduct the desired task. In most cases, the goal is to safely move the robot to a target position to perform a deed. The action of determining the sequence of steps to do this is called path planning. For a path planning algorithm to be effective, it must consider numerous parameters: reaching a target position and avoiding nearby obstacles.

There has been a lot of research performed on path planning for mobile robots. Many researchers have proposed very efficient methods. However, in the case of manipulator arms, the structure of the robot adds to the complexity of the solution. Although many approaches have been put forth, there are none that can truly be called robust where the algorithm is independent of the robot configuration and works in an arbitrary environment situation.

1.1. Objectives

The goal of this research is to explore ways to develop a robust solution for collision-free path planning of robot manipulators. The proposed algorithm is designed to work in a cluttered environment for an arbitrary manipulator configuration while keeping computation time small such that the advantages of automating the process are not compromised by a heavy computational workload.

Manipulator control becomes considerably more difficult as the manipulator increases in complexity. For more than a few degrees of freedom, direct computation of the robot

parameters becomes an extremely complex task, sometimes without an analytical solution. As a result, manipulators are often designed to facilitate this computation at the expense of having robots that are not optimized for the task at hand. One aspect of the thesis examines approaches to compute the robot parameters regardless of the complexity of its design.

After an investigation and a comparison of various classical modeling schemes in the context of robot path planning, the framework in which the research is performed assumes that environment information is gathered from probabilistic datasets. This type of model provides rich information regarding the workspace and contributes to reduce computation as uncertainty is already encoded in the dataset. The mapping of the environment is encoded in a sequence equivalent to tree structures, called quadtree in 2D, octree in 3D or n-tree in a general case. This sequence is a compact format that considerably reduces memory requirements and neighbor searches.

Based on the modelling information acquired separately, the solution that is proposed and evaluated in the present work consists of having the path of the end effector of the robot computed with the use of attractive and repulsive potential fields. These fields attract the effector towards the goal position and simultaneously push it along with the rest of the structure away from obstacles.

To ensure that the rest of the robot structure safely travels through the free space, an original design of a fuzzy logic controller is introduced. This controller interprets the configuration of the robot and the surrounding repulsive obstacle forces and determines a safe trajectory for all the joints such that the entire structure reaches the target configuration defined during the phase of path planning for the end effector. A general diagram of the steps followed is shown in Figure 1.

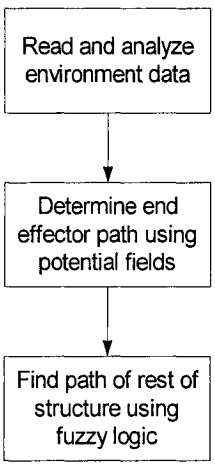


Figure 1: Block diagram of the proposed path planning approach

1.2. Contribution

The contribution of this research is to provide a robust and adaptive solution for manipulator path planning. The use of probabilistic datasets and an original encoding scheme allow for a compact yet accurate representation of the environment that minimizes the computational effort for the search of neighbor configurations required to define a trajectory. The use of attractive and repulsive potential fields offers a simple method for global path planning. This classical approach is combined with a fuzzy logic controller that combines the manipulator configuration and obstacle information to determine a safe, collision-free path for the whole structure of the robot arm. The proposed setup offers the flexibility to use an arbitrary manipulator for a given environment while avoiding the extensive computation related with manipulator control based on classical inverse kinematics. Each component of the proposed path planning approach is experimentally tested and analyzed with various environments and robot configurations.

1.3. Structure

The thesis is structured as follows: Chapter 2 provides a review of literature of the main concepts in the field. It is primarily aimed at recent documents starting from 1995. In Chapter 3, the classical occupancy grid encoding approaches and the neighbor cell finding algorithms are detailed and experimentally compared. The grid representation of the robot

working environment is put to advantage to minimize the number of neighbors to find when determining a safe path. An original encoding scheme is presented and compared against other approaches found in the literature. The two phases of path planning are presented in Chapter 4. A global approach guides the effector towards the target position using potential fields, while a local algorithm utilizes fuzzy logic to overcome the complexities of manipulator modeling and inverse kinematics to guide the rest of the structure around obstacles. Experimental results are shown and analyzed in Chapter 5. Finally, the conclusion provides an overview of the concepts introduced in this thesis and explains future research directions and potential uses for the proposed approach.

CHAPTER 2: REVIEW OF LITERATURE

2.1. Introduction

This chapter offers a survey of the recent advances in the field of robotic path planning and collision avoidance and an introduction to other concepts used in this thesis. It is divided into path planning, potential fields, probabilistic representation, encoding grids and manipulator control.

2.2. Path planning

Avoiding collisions during the movement of a robot is a critical issue that implies efficient path planning strategies. Many researchers have proposed different path planning approaches for mobile robots that yield efficient results. Unfortunately, for manipulators, the process is more complex as the structure of the manipulator introduces difficult situations that do not appear for mobile robots. Therefore the majority of classical path planning techniques cannot be directly transposed to manipulator robots.

However, there are a few strategies that have been inspired by mobile robots path planning technologies and adapted to manipulators. The general trend consists of dividing the path planning strategy into two steps: first, a global approach determines the path of the end effector in the free space (this is similar to path planning for mobile robots as the rest of the structure is not considered), and then a local path planning method determines the configuration for the whole robot structure given the desired positions of the end effector in the environment along the trajectory.

2.2.1. Overview of literature

Although path planning is an essential step in robot automation, many of the approaches used are simple heuristics: successively searching for face connected cells or configuration with or without the minimization of a criterion. These usually provide limited results depending on the complexity and configuration of the environment. These techniques are

applied to both mobile robots and manipulator arms. In an attempt to improve their performance and robustness, numerous alternatives to those methods have been proposed.

Lee and Kardaras [16, 17] present a different path planning algorithm for mobile robots that makes use of "via points" (denoted VP) that connect to form the path of the robot. The number of points varies according to the complexity of the surrounding environment: the path along a straight line or in free space will generally require very little number of VP, while the paths closest to an object, which usually tends to "curl" the path, will introduce numerous VP. Simulated annealing is used to smooth out the resulting trajectory. The use of a neural network properly trained is also proposed to make the entire process more parallel.

Nam, Lee and Ko [21] present an approach for collision avoidance of moving objects. The proposed approach recalculates the potential field only for a small region known as the accessible sweep region, which reduces the complexity of rebuilding the entire potential field. The authors use the knowledge of the position, speed and acceleration of the objects to move the mobile robot. However, this requires an enormous amount of calculation, as numerous measurements need to be taken at very short intervals. From the speed and acceleration data obtained, a form of regression is used to find the most adequate solution using heuristic approaches. The approach is demonstrated to work mainly on simplified environments where distances are large; therefore the sampling time is larger.

In [23] by Nishimura *et al.*, potential fields are used in conjunction with genetic algorithms for path planning for a manipulator arm. The potential fields are used to guide the end effector, while the genetic algorithm ensures collision avoidance for the rest of the structure of the robot. The path planning sequence is not very well defined but the paper presents a very good overview of generic algorithms in the context of robot path planning. The variations in joint angles are coded as genes. This gene is then passed through a series of genetic operations: genetic coding, fitness, crossover, mutation, natural selection and parameter tuning. These operations mutate the gene to a pseudo-random position that is evaluated. These operations are stopped when the robot has reached the desired

configuration. The results reported confirm the effectiveness of the proposed solution. There are a few drawbacks to this approach that can be observed, most notably is that the algorithm finds a solution through pseudo-random means. Even though the randomness is controlled, the authors do not include any estimation of execution time, which leads to the deduction that this approach may be very lengthy.

Solteiro Pires and Tenreiro Machado [33] also present an approach to collision avoidance using a standard genetic algorithm. Their concept is to minimize a penalty function that represents the configuration of the robot manipulator (obstacles, robot position, angular speeds...) by using the fundamental functions of genetic algorithms (reproduction, crossover and mutation). The approach used is simplified and test environments are simplistic. However, the authors show the ability to avoid collisions in path planning.

In [24], Oriolo *et al.* present a heuristic-like approach to path planning where the end-effector path is given and must be followed in a tracking operation. The algorithm presented by the authors determines the position of the structure of the robot arm throughout the given path. This path is segmented into smaller steps where the robot configuration must be computed. Every possible configuration of the robot is analyzed until one is found that does not create collisions with the environment. Those solutions are found in a random order and do not guarantee the optimal solution in terms of joint displacement and computation time.

Lin and Chuang [20] offer a different perspective on manipulator path planning. The authors propose to use guide planes (GP) as intermediate goals in the 3D workspace. They provide a general direction for the manipulator to move forward towards the goal by specifying polygons that represent regions between the obstacles. Using continuous repulsive fields, the algorithm finds the path with the lowest value for repulsion within the boundaries of the GP. This method yields good results; however, the authors assume "that the sequence of GPs is given in advance". The generation of the GP is not precisely

defined. By analysis, this approach seems to avoid obstacles when they are far from each other, but may lead to some problems when it is not the case.

Hwang *et al.* [11] present an interesting approach for path planning with mobile robots that relies on adapted environment representations. The authors observe that the principles of quadtrees (and octrees) possess two intrinsic drawbacks: the quadtree representation needs to be of high resolution to model the obstacles correctly and the quadtree method cannot guarantee to find the shortest path due to large cells that tend to appear in free space. Instead of using square cells in the quadtree sequence, a 2D environment is converted into a triangular grid and blown into 3D where the slope of the triangle represents the proximity of an object. Free space will have a zero "z" component while obstacles will be nonzero. The resulting grid is then simplified by removing some vertices representing the edges of the triangles; this is done to reduce the size of the mesh by exploiting the redundancy of regions of similar occupancy. The path planning strategy is not extensively detailed, but it uses forces to push the robot and generate a path using a heuristic approach. Path planning results are however good, as the approach reduces the computation time by a factor of 10 (over standard quadtrees). On the other hand, the mesh computation is very lengthy (50 sec compared to 0.09 sec for path planning). No extension is proposed for an equivalent mapping for 3D environments.

In [19], Lian *et al.* present a simple approach in computing a collision-free path for a manipulator arm. They use repulsive forces to push the robot away from obstacles. However, their approach does not specify the inverse kinematics that the authors used even though they hint towards neural networks and fuzzy logic. Repulsive forces are classified into three different categories called "obstacle patterns" from which the force parameters are extracted. They are shown in Figure 2.

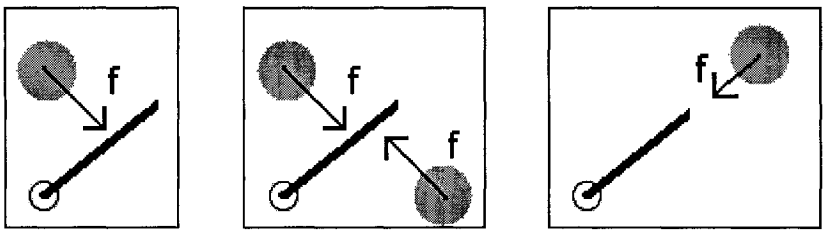


Figure 2: Repulsive forces situations

Collision avoidance commands are extracted depending on the obstacle pattern and the key obstacle, which is not clearly defined. The example provided is simplistic and the combination of local and global command generation is simple but not very well defined. Joint movement is determined by the probability of collision for every joint.

Ando [1] presents an algorithm to determine the path of a manipulator arm through a crowded environment. A general path is found using global path planning methods and the algorithm then finds sub-goals throughout the path that satisfy this global movement. These sub-goals are found using a general A* algorithm. If this search does not yield an adequate position, the global search is called again and the global path is recomputed using a different A* range parameter. The approach aims to reduce the computation times. Unfortunately, the strategy does not seem to encompass general robot architectures. Even though the strategy is aimed towards manipulator arms, no consideration is made of the inverse kinematics. In fact, the results presented only deal with the path of the end effector.

Probabilistic Roadmaps (PRM) is another relatively new approach to robot path planning that is presented by Leven and Hutchison in [18]. Path planning is again done in two steps. In the first step (performed offline), a series of random configurations in the configuration space (C-space) are found and are called nodes. These nodes are then connected using a local path planner. The second stage links those nodes from the start position to the goal position. The roadmap that is obtained indicates the probability in a region where a safe robot configuration may exist. The limitation of this approach is the large number of samples that are needed, up to ten million, according to the authors. This is the main focus of research for PRM. In [18], a sampling approach is proposed that is based on

manipulability or dexterity, an intrinsic property of manipulator arms. The authors stipulate that a lower number of sampling is needed when the manipulability of the robot is high, such as in free space. In opposition, a larger number of nodes must be sampled where the range of motion is restricted, for example near obstacles or joint limits. PRM are an interesting approach to local path planning methods but require extensive computation, especially during the pre-processing stage as a random search is performed. The methods introduced in [18] offer an improvement of 2-3 times over standard PRM.

2.2.2. *General observations on existing path planning techniques*

The previous approaches offer a diversified perspective to the problem of path planning. However, it is difficult to find a general solution that is robust for an arbitrary configuration of the manipulator and the environment. Often, the focus is put on mobile robots and the complexities introduced by manipulators are avoided. Attempts are made to reduce the online computation at the expense of enormous offline and setup times, which is not a valid approach for most robotic systems. Unfortunately, we must conclude that the literature does not yet provide a robust solution that would accommodate a large set of environment configurations, especially if manipulator arms are of concern. Therefore, general solutions must still be investigated. For that matter, a combination of some of the most promising approaches might be explored, as discussed in the following sections.

2.3. Potential fields

Potential fields have taken an important place over the last decade in robot path planning research as they offer a very interesting approach that is relatively simple and efficient. The overall potential field is made up of an attractive field, which attracts the robot towards its objective, and a repulsive field that pushes it away from the obstacles. As discussed previously, many of the path planning approaches still make use of simple heuristic algorithms, which may yield a quick result but are usually not optimized or require enormous computation. When not designed from a general enough perspective, an important drawback of potential fields is that the robot (whether a manipulator arm or a

mobile robot) tends to get trapped in what are called local minima. These minima in the potential surface prevent the robot from reaching its destination. Efforts to overcome this problem represent a large part of the research regarding potential fields and are presented in this section.

2.3.1. Concepts

Originally proposed by Khatib [12], potential fields have been refined for a number of years and used for many applications including path planning for robots. They offer a simple yet efficient method to encode the location of obstacles in a given environment through a representation that can be directly interpreted by classical path planning techniques. A simple analogy to potential fields could be to picture a marble rolling down a hill and letting it reach the lowest point of a valley. Provided that this point in the valley corresponds to the target configuration to reach, a trajectory can be computed for the robot by following the negative gradient of the surface. This section summarizes the main techniques of potential fields.

2.3.1.1. Discrete repulsive fields

The repulsive potential fields result from the presence of obstacles in the environment. Their purpose is to repel the robot away from these obstacles. These fields are generally computed as a function of the distance from objects boundaries. As the object most likely to create a collision is always the closest one, only that distance is usually taken into consideration.

A safety margin may be introduced around the objects to account for the structure of the robot and the uncertainty of sensors to measure an obstacle's location. In a typical application, the distance to the nearest obstacle is first computed for every point in the environment. The resulting array is an amalgamation of distances. By design, all immediate neighbors of a point are of similar distance, also, the distance of a point to the closest obstacle can be, at most, one larger than the smallest of its neighbors; it can also be only one smaller. For example, the neighboring points to the obstacle edges are assigned a distance of

1, and then the neighbors to those points are assigned a distance of 2, etc... For a Cartesian representation, made of a discretization of space in a grid of cells of a given resolution as shown in Figure 3, every point can have either 4 or 8 neighbors, depending on whether diagonal neighbors are considered to be adjacent cells or not. When extended to 3D space representation, 4-neighbor mapping results in 6 neighbors, and 8-neighbor mapping leads to 26 neighbor cells. In this research, we have chosen to use the representation with 8 neighbors in 2D and 26 neighbors in 3D as it increases the flexibility of path planning in cluttered space.

4	3	2	3	4
3	2	1	2	3
2	1	0	1	2
3	2	1	2	3
4	3	2	3	4

a) 4 neighbors

2	2	2	2	2
2	1	1	1	2
2	1	0	1	2
2	1	1	1	2
2	2	2	2	2

b) 8 neighbors

Figure 3: Neighbor mapping in a grid-based representation

The table of distances for a sample environment is given in Figure 4 where the black squares represent obstacles while the number inside the white cells represents their respective distance to the nearest object. In order to consider the lack of measurements taken beyond the edge of the environment, it is usually assumed that outside the workspace is an unknown space and therefore may contain obstacles. Therefore, the border of the workspace is treated as an extra obstacle. As a result, cells are assigned a distance to the nearest object or border as shown in Figure 4.

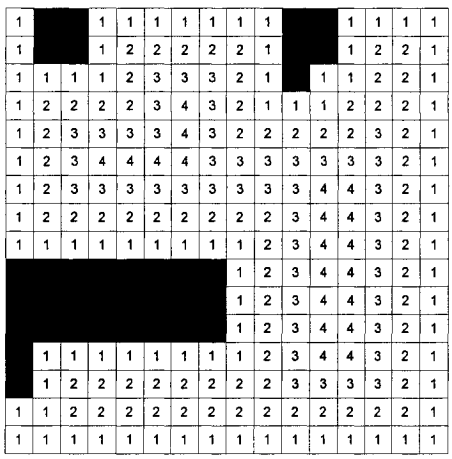


Figure 4: Sample table of distances

Once the table of distances is obtained, the repulsive field can be directly computed as an inversely proportional function to the distance for each cell. That is, a large distance means that the position is far from an obstacle and the repulsive field is weak. Inversely, if the distance is small, then the presence of the obstacle will cause a large repulsive field.

There is an important problem that plagues the computation of tables of distances and repulsive fields. As they rely on a computation of distance to the closest object, a large amount of computation is required to determine the successive neighbors of every point in the environment.

2.3.1.2. Free space

In order to determine a collision-free trajectory, the safe regions exempt of obstacle must be found. This subspace is called the free space. Starting from the table of distances, the computation of the free space is straightforward. A threshold representing the desired safety margin around obstacles is applied to the table of distances to account for sensor errors on the precise location of obstacles. The resulting array is segmented in two components: one consists of all cells where the distance to the nearest object is larger than the security margin while the second is made of cells with a distance smaller or equal to the security margin, which includes the objects.

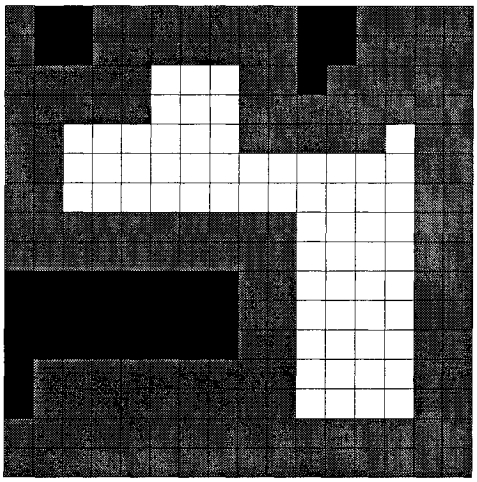


Figure 5: Sample free space

Following the example shown in Figure 4, a security margin of 2 is used to find the free space, as depicted in Figure 5. The resulting free space is represented in white cells in which the robot can safely traverse while the black squares represent obstacles and the security margin is shown in grey.

2.3.1.3. Discrete attractive fields

The attractive field is meant to progressively guide the robot from its current configuration to the target. To achieve this, an attractive field can be built that represents the distance from a position in the workspace to the target. It is computed from the free space obtained from the distance table and consists of tagging each cell with a discrete distance value computed from the goal configuration in number of cells. This new distance does not refer to that contained in the table of distances that is defined with respect to obstacle surfaces. The new distance values are found by means of a wave propagation originating from the target position and limited to the predefined free space. Every neighboring point to the target position (denoted as "T") in the free space is assigned a distance of 1, then the neighboring cells to those are given a distance of 2, and so on, until all the cells in the free space have been tagged. The resulting discrete attractive field is shown in Figure 6.

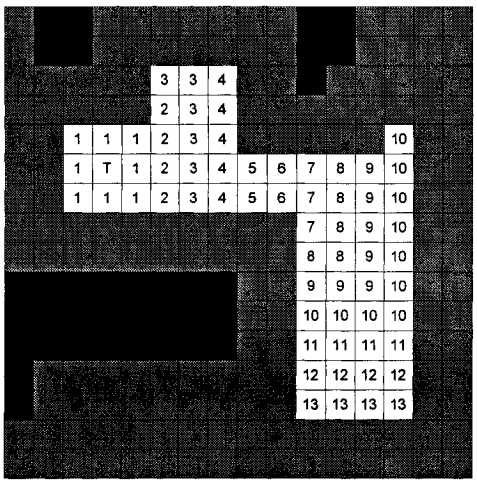


Figure 6: Sample attractive field

Ideally, guiding a robot through this attractive field by following the down slope should make it reach the objective. The output path can be calculated step-by-step with successively finding the neighbor cell with the smallest distance to the target and following the "downslope". Figure 7 displays a greyscale mapping of the discrete attractive field for a larger environment with numerous obstacles depicted by white areas. The target position is, in this example, denoted by "T" near the top-left corner. The objective is to bring the robot from an arbitrary start position located in free space by progressively moving towards lighter pixels and eventually to the target position. Two sample trajectories are shown which start respectively from two initials configurations: near the lower-right and the upper-right corners of the map.

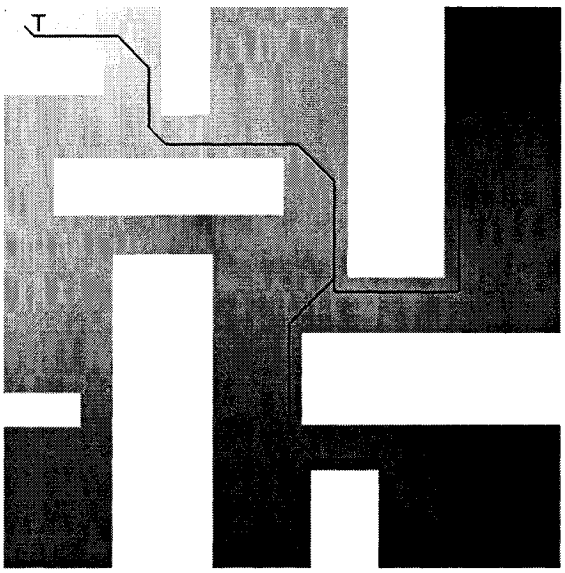


Figure 7: Example of an attractive field

Even though such a straightforward procedure has great chances of success for a point-like mobile robot, it is rarely successful for a manipulator arm as repulsive forces exert important pressure on the totality of the structure. As a result, the arm finds itself in the presence of contradictory forces originating from repulsive and attractive fields and tends to get trapped in local minima. Although the use of discrete potential fields reduces this effect, robust solutions have not yet been identified to eliminate this problem when considering manipulator arms, as there are many points of interaction distributed along the robot structure. These situations arise because the entire structure feels different forces that contradict its movement toward the goal. In this research, an original combination of potential fields for global path planning and fuzzy logic for manipulator control and local path tracking is proposed to overcome some of these limitations.

2.3.2. Overview of the literature

Potential fields (also known as artificial potential fields) are a promising approach for path planning. The concept has enjoyed popularity with its relative simplicity and speed of execution. Many researchers have proposed algorithms to reduce or to circumnavigate

around the typical problems inherent to potential fields. Here is an overview of the main research directions.

2.3.2.1. Detecting and escaping local minima

Barraquand and Latombe [2] have proposed a classical approach for path planning. The authors use a potential-field based method that follows the gradient descent to approach the target position. If the robot becomes stuck in a local minimum, the algorithm tries to escape by the addition of random movement widely known as the Monte-Carlo approach. The process is repeated until the path reaches the target position. The proposed approach shows good results in dealing with manipulators with a large number of degrees of freedom, however, the approach is very lengthy when dealing with narrow corridors. The random search for a valid solution leads to non-uniform planning times and is not repetitive; this represents a major drawback for real-time applications as required in modern robotics.

Caselli *et al.* [4, 5] present a method for escaping local minima. The approach determines if the robot is currently located in a local minimum and uses different methods to escape it. The first approach (Straight Line) is to move the robot in a random "up-hill" direction until a criterion is met. The second algorithm (Straight Line Select) eliminates unwanted candidate directions therefore optimizing the escaping path and minimizing the occurrence of the robot falling in the same minimum. The results obtained are not very suitable for real-time applications; however, the simplicity of the approach is interesting.

Park *et al.* [25] propose a similar approach where once the robot is trapped in a local minimum, a random solution is found using simulated annealing from the set of neighbors to the current solution. The simulated annealing method searches for random robot configurations and evaluates them based on an optimization criterion that attempts to free the robot. As the Monte-Carlo approach, this random search method will not lead to uniform results, which limits its use for real-time application. The authors present very good results for mobile robots, but do not mention an extension to manipulator arms.

Most approaches assume that the robot has a fairly simple shape. However, in order to be more rigorous, a more precise representation can be used, but it introduces different problems. Chang [6] examines the fact that every point in an object will exert a different force to every point on the robot. Hence different forces are applied to different parts of the robot structure, creating many possible attractive and repulsive fields. The method proposed by the author searches through the fields until a valid path can be found. The trajectories found using the proposed algorithm are very interesting and demonstrate that the approach is able to fully model the robot and determine a safe global trajectory. However, the algorithm requires significant amount of memory to store the robot representation and environment information, therefore leading to lengthy computation times.

Chengqing *et al.* [7] present an interesting method to detect local minima. They observe that minima are created most of the time by concave objects, or a series of convex objects forming together a concave object in the workspace. The proposed method is used when the robot is trapped in the local minima as it tries to determine the optimal direction to escape it by determining the largest opening of the concave object. The approach used is interesting, however, real-time application of the proposed approach is not demonstrated.

2.3.2.2. Other representations of potential fields

A number of different approaches to improve the efficiency of potential fields by slightly changing their representation have been developed. While some try to decrease the generation time in order to deal with evolving environments, others use different physical phenomena that mimic potential fields such as heat transfer or circulatory fields.

A limiting factor to using potential fields for path planning is that its generation can be somewhat cumbersome especially for evolving environments. Piaggio and Sgorbissa [29] propose a method to statistically reduce the potential field calculation. Their approach is to divide an area near to the robot or end effector into circular sectors of equal width and into equally spaced rings. The resulting grid will have a similar shape to that of a cylindrical coordinate system. The sector explored is updated beginning with the smallest radius where

an object is found. The approach has shown a reduction of 11% in computation time over traditional grids. However, the technique implies that a robot setup is designed such that proximity sensors are mounted on-board a mobile robot.

An interesting adaptation of potential fields is presented by Sing, Stephanou and Wen [32] in which an artificial current is assumed in the obstacles instead of the typical electrostatic charges. The effect of this current is to create a circulatory field that reorients the velocity of the robot, guiding it around the obstacle. The force exerted by all the obstacles depends on the velocity of the robot as well as on the distance from the objects and is defined as:

$$\vec{F}_{obst} = \dot{x} \times \vec{B} \tag{1}$$

where $\dot{x}$ is the speed of the robot and $\vec{B}$ is the combined field of all the elements, which is computed as:

$$\vec{B}_i = \int \frac{K_i \times \hat{\dot{x}}}{r^2} da \tag{2}$$

where K_i represents the surface current, r is the distance to the object and a is the area affected by the force. The overall field B is the vectorial sum of all B_i. The publication does not provide any indication about computational times.

Along the same line of ideas, an appealing approach of potential fields is presented by Wang and Chirikjian [36], where the potential field is not based upon magnetic fields but upon heat transfer. A collision-free path is found by finding the path of least thermal resistance. Although the process looks similar to regular potential fields and no feasible improvements can be revealed, the authors state that "the advantages of using variable thermal conductivity are that it allows for a simple geometrical domain regardless of obstacle complexity and can handle changes in the environment".

A very simple, although not optimized, method for escaping local minima is presented by Yun and Tan [37]. It consists of using common path planning algorithms and a wall-

following algorithm when the mobile robot is trapped in a local minimum. The wall-following algorithm contains two steps:

1) Once the robot is trapped in a local minimum, it arbitrarily follows the wall at a random direction.

2) The robot will continue to follow the wall until it reaches a decreasing potential field flow. At this point, the robot will switch from the wall-following control to the potential-field control.

The method presented is very simple and works in many situations. It is very easy to bring the process real-time; however, the resulting path is not optimized and no mention is made of manipulator arms. This publication demonstrates that development of robust path planning solutions is still in its infancy.

2.3.2.3. Modeling of potential fields

A different representation of either the workspace or the robot can be useful for many different reasons. Complex objects can be modeled as simple ones or using a more memory-efficient manner. Modeling robots of complex shapes especially highly redundant manipulator arms can drastically reduce the efficiency of an approach. An efficient model must therefore be used to reduce the complexity of the algorithm.

Conkur and Buckingham [8] examine highly redundant manipulators and use them in very crowded environments. In order to speed up the process, the obstacles are modeled as ellipses with a security margin while the links of the robots are modeled as lines. The approach exploits this elementary representation to avoid collisions. Also, since the robot used is a highly redundant manipulator arm, the interaction between the links must also be taken into consideration. Some very interesting results are presented and the online computation process, consisting mainly of the path planning method, is generally fast.

The purpose of the approach used by Kitamura *et al.* [13] is to use the quadtree representation to find the best path for the robot to follow. Although the path planning algorithm consists only of heuristic techniques, the quadtree approach shows considerable

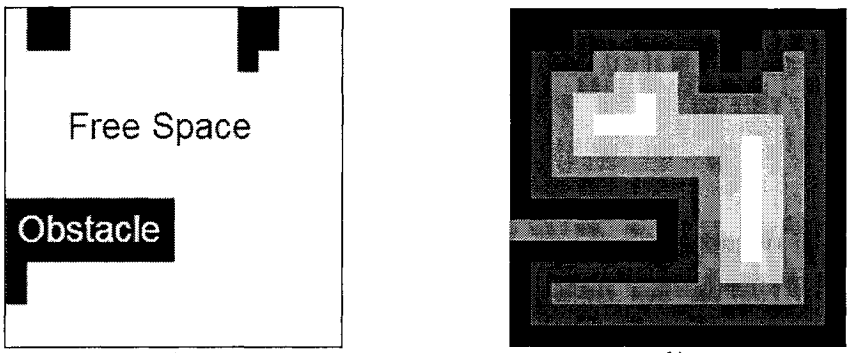
improvement over the regular grid-like representation. The current approach reduces the number of nodes to explore in order to determine the best path; the results obtained show an improvement of up to three times. The paper also presents a good application for mobile robots since rotations are also calculated. The authors present some theory with regards to moving obstacles but this aspect is not extensively detailed.

Laliberté and Gosselin [14] have proposed an interesting method that reduces the occurrence of local minima. The potential fields are discretized and the attractive field is computed by means of wave propagation from the target position. This is an attractive and simple method that can be easily used for path planning. It has been validated on redundant 2D and 3D manipulator arms but demonstrates limitations when the robots try to reach behind obstacles. The proposed approach relies on the analytical solution to the inverse kinematics of the robot, which increases complexity and limits the generality of the solution.

2.4. Probabilistic representation

2.4.1. Overview

Since the representation of the working environment is critical for the success of a robotic application, selection of the modeling scheme for the workspace is another important factor in developing robust path planning strategies. Even as the most accurate range sensors are often error-prone, obtaining many samples of the same data point and taking into account the uncertainty associated with each measurement is important for the robustness of the environment model. Inspired by a 2D approach introduced by Elfes [9], Payeur *et al.* [28] have proposed a method of 3D data fusion where a probability (with 0.0 representing empty space and 1.0 corresponding to occupied space) is used to encode the occupancy of a given cell in an occupancy grid, as opposed to discrete occupancy levels (occupied, empty or unknown). This probability is based on the number of measurements available and the fusion of those readings. A probabilistic model of a simple 2D environment is shown in Figure 8. For clarity, only 5 levels of probability are used and mapped as grey levels ranging from black (occupied space) to white (empty space).



a) b)

Figure 8: Probabilistic representation a) obstacles location, b) probabilistic occupancy mapping

Following this representation, the occupancy state of cells located inside obstacles is unknown and given a probability of 50% (depicted with the middle shade of grey). Outward from the obstacle edges, the probability progressively lowers as the certainty that cells are empty increases. This introduces "fuzziness" around obstacles to represent the uncertainty of the readings, as shown in Figure 9 for a more complex environment. In the latter example, dark regions are associated with the surfaces of the objects and the boundaries of the environment which are also considered as obstacles to prevent the robot from leaving the modeled workspace.

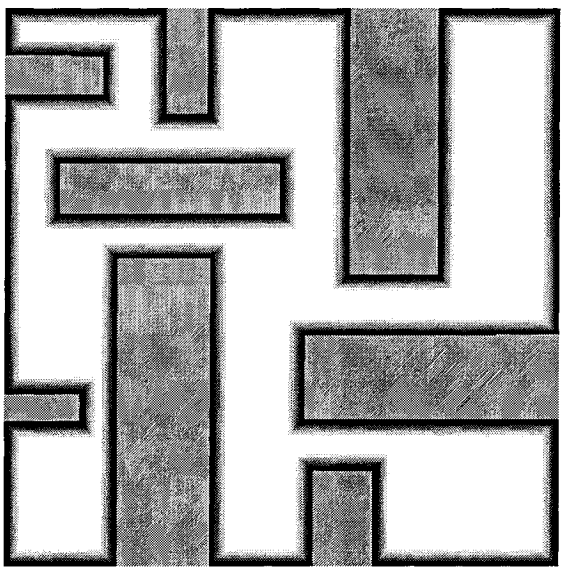


Figure 9: Example of a probabilistic environment mapping

The introduction of probabilistic datasets in replacement of deterministic ones mainly influences the computation of repulsive potential fields and the determination of free space as demonstrated in [27]. In environments where narrow corridors have to be traversed, the richness of a probabilistic representation provides knowledge on the reliability of the environment model. In a deterministic representation, this uncertainty can at best be encoded into fixed safety margins around each obstacle and therefore does not provide any flexibility to the path planning operation.

2.4.2. *Choice of a representation*

After having investigated various mapping strategies, numerous factors led to choosing probabilistic datasets for the present work. With the richness of the information that it provides, there is a direct relationship with the repulsive field. Instead of computing the distances that make up the repulsive fields with classical representations, this information is found intrinsically within the probabilistic model of the environment. A close analysis of the advantages of using probabilistic datasets was performed in an exploratory phase of this research [34], and demonstrated that a 15% improvement in computational workload can be achieved by using probabilistic encoding instead of using conventional environment representations. These results were found in path planning execution times for a mobile robot application and a global path planning strategy. In its use in global path planning, the introduction of probabilistic datasets clearly reduces the computational effort for determination of the repulsive potential field. We conclude, therefore, that the use of probabilistic datasets is justified to pursue our experimentation.

2.5. Manipulator control

As mentioned before, although many researchers have proposed approaches for path planning for mobile robots, manipulators have not been the focus of as much research. This fact comes mainly from the difficulty of controlling the entire structure of a manipulator which moves and interacts with itself and the surrounding environment as well as the inherent complexity of the kinematic model. Moreover, most of the papers in the literature

only present research in controlled environments and are generally not very suitable for an arbitrary robot configuration or environment. As the goal of this thesis is to explore methods that can be applied to generic manipulator setups and provide a collision-free path in cluttered environments, an examination of the literature that deals with manipulators control with path planning applications has been performed. In this thesis, starting from a global path for the end effector found through the use of potential fields, a path tracking method is developed to determine the sequence of robot configurations that follow this trajectory. A fuzzy logic approach is considered as it can handle the non-linear problems inherent to the analytical solution of the inverse kinematics of a manipulator arm that would be very difficult or impossible to solve using traditional or analytical methods. Research work investigating similar solutions found in the literature is summarized in the following section.

2.5.1. Review of literature

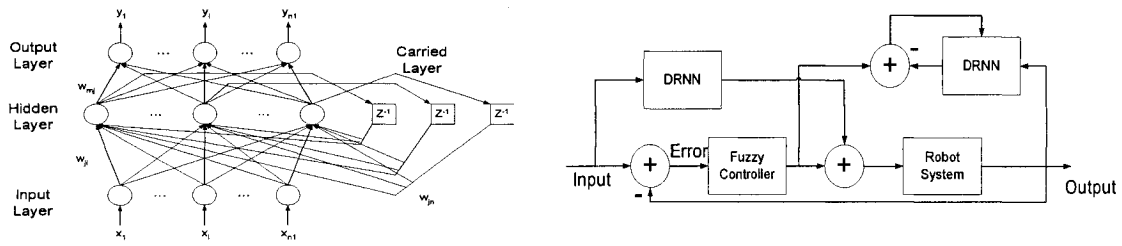
Nedungadi [22] proposes an interesting approach to manipulator control. By introducing fuzzy logic as the robot controller, the author tries to overcome the long and arduous computation of the inverse kinematic solution. The author uses end effector velocity and position change as fuzzy inputs. Both are then encoded into fuzzy form and evaluated according to the rule bases developed by the author. This approach is simple and allows extension to many different configurations of robots such as lengths of joints, number of joints, 2D or 3D spaces. The goal of the algorithm is to track a pre-defined end-effector path using the proposed fuzzy logic controller. The work is limited to an empty environment. The results reported present small error in tracking and fast computation, but they do not consider the effects of obstacles, which is critical.

Beheshti and Tehrani [3] propose an algorithm for manipulator control with the use of fuzzy logic. An adaptive fuzzy logic (AFL) approach proposed in previous research is used for solving the inverse kinematics problem with a collision avoidance method that is introduced to complete the generality of the approach. The authors propose that the manipulator travels the workspace until it enters a threshold region defined around

obstacles. At this point, the fuzzy rule base is changed to cause the manipulator to leave the region. This approach is simple and is well adaptable to any manipulator, however, since the obstacle information is not directly included as a fuzzy input, the algorithm is less reliable to determine a safe and optimized path and can lead to lengthy computation times.

Ham *et al.* [10] propose another approach for fuzzy logic controllers that guarantee global stability and performance. They consider a number of intrinsic robot parameters: centripetal and Coriolis effects, gravity effects, static and dynamic friction and disturbance due to load variation and/or modeling errors. The authors show that the position error of the joints can converge towards zero by defining the fuzzy parameters according to guidelines that they state. The fuzzy logic approach proposed by the authors appears to be slightly better than a nonlinear robust control. However, the algorithm implementation is very lengthy and would not be suitable for real-time applications.

Tian and Mao [35] propose a combination of fuzzy logic and neural networks for the inverse kinematics of manipulator arms. The fuzzy controller is used as a feedback to provide control signals for the manipulator arm. In the feed-forward configuration, a dynamic recurrent neural network (DRNN) is used to model the inverse dynamics of the manipulator system. This setup offers an increase in performance and the stability of the control system. Figure 10 shows the architecture of the network and block diagram of the system. The neural network includes a hidden layer that consists of non-linear neurons and an output layer consisting of linear neurons. The non-linear neurons are used to capture the non-linearity of the input function, such as the inverse kinematics of the robot. The linear neurons allow the approximation of the input function within an arbitrary range. The fuzzy logic controller is used to model the robot arm dynamics.



a) Architecture of DRNN

b) Block diagram of control system

Figure 10: DRNN architecture and block diagram

2.5.2. *Choice of an approach*

The approach brought forth by Nedungadi [22] offers a simple method to control a manipulator arm. Since the fuzzy logic controller only requires the forward kinematics, this approach is well suited for use with any robot. The proposed method can also be extended to include the repulsive forces of nearby obstacles. Building on the ideas proposed by Nedungadi, the research work presented in this thesis proposes a generalization of the fuzzy logic approach and a combination with potential field strategies discussed previously. In order to achieve a generic solution, 3D environments and the presence of objects to avoid in the manipulator workspace are taken into account.

2.6. Conclusion

Many researchers have proposed approaches for path planning for mobile robots, but manipulators have not been the focus of as much research. The algorithm brought forth by Nedundadi [22] offers a simple method to control a manipulator arm. Through improvements of this approach, this work proposes a strategy that combines the forward kinematics of the manipulator and obstacle information so that it is robust and can be used for arbitrary manipulator architectures in cluttered environments.

CHAPTER 3: OCCUPANCY GRIDS AND NEIGHBOR SEARCH

3.1. Introduction

The occupancy information for an environment can lead to a large amount of data. Processing this data can result in extensive computing demands. However, cluster regions of similar occupancy state can exist and be regrouped to reduce the amount of information that needs to be encoded and analyzed. By regrouping these regions, it is possible to condense the occupancy information since large regions are treated as a single area. The concept is to reduce the resolution needed when the occupancy of a region is consistent while keeping a high resolution when a cluster is not uniform. Multi-resolution cells have been used to simplify environments in 2D space with quadtrees, and octrees defined in 3D space.

In this research, the free space is analyzed rigorously: it is given a distance to the target position, and it is explored to determine the optimal trajectory. The need for a compact and efficient representation is therefore critical. The robot needs to travel by following a sequence of cells in the environment: from the current position, a series of waypoints must be found until the target position is reached. The relationship between those two cells needs to be determined efficiently by neighbor searching algorithms. By using multi-resolution cells, large sections of free space can be analyzed and interpreted in a single step while avoiding unnecessary computation and searching.

With the use of these multi-resolution grids, we are able to take advantage of the compactness of the model therefore reducing memory requirements for the application and allow fast neighbor searching by greatly reducing the number of neighbor cells that have to be analyzed.

This chapter presents an original scheme for multi-resolution occupancy grids encoding that reduces the memory requirements to map the working environment. Also, an efficient

neighbor search algorithm taking advantage of the proposed encoding scheme is introduced. Finally, this original approach is compared with techniques found in the literature.

3.2. Encoding scheme

3.2.1. Multi-resolution occupancy grids

The purpose of using multi-resolution grids is to regroup regions of similar occupancy state and by consequence reduce the number of cells that have to be processed. A multi-resolution grid can be encoded into an equivalent tree-like structure containing occupancy state of different regions that define a working environment of a given size. At a given resolution level, a cell is marked either entirely free, entirely occupied or a combination of both (unknown, inconsistent or non-uniform). Unknown regions are combinations of free and occupied regions. Those cells are subdivided and further analyzed until a consistent occupancy state is reached for each component. For simplicity, the approach is depicted for standard deterministic maps but can be generalized to probabilistic models.

The encoding sequence follows a tree-like structure: the cell at the top of the tree represents the occupancy state of the entire region. Unless the region is entirely empty or occupied, this cell has an unknown occupancy. It is thereafter divided into 4 quadrants (or 8 octants in 3D), which are analyzed for consistency of their occupancy. If they are consistent, they will be assigned that occupancy value, however, if it is not, then that current cell is further divided into quadrants. This is done until the entire environment is mapped at the desired resolution level. Figure 11 illustrates an example of the different levels of a quadtree. In this figure, the free regions are indicated with white cells, the occupied regions are in black and cells of unknown state, which need further subdivisions, are in grey.

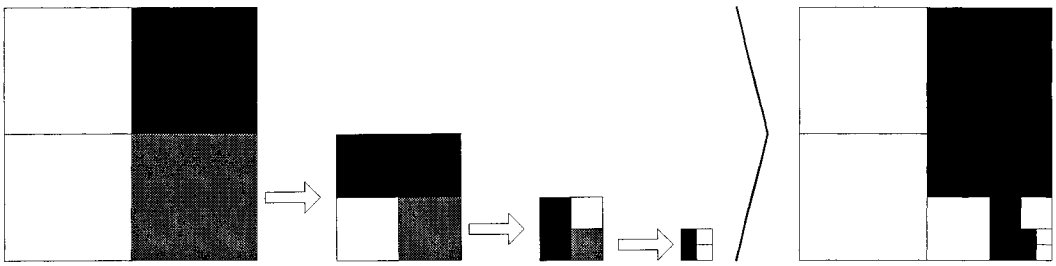


Figure 11: Quadtree decomposition

From the previous example, the multi-resolution grid can be stored in a tree format. The corresponding tree structure is shown in Figure 12. As this model maps a 2D space, each inconsistent cell is divided into 4 quadrants using the numbering convention shown in Figure 13.

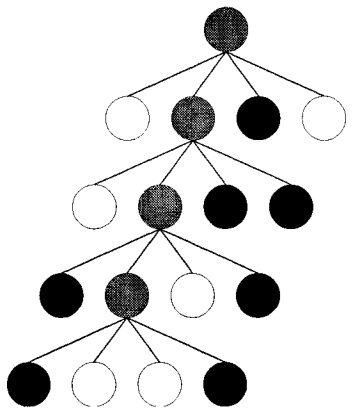


Figure 12: 2D tree structure representation

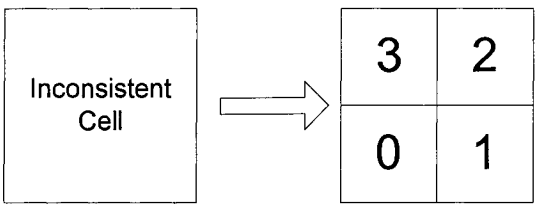


Figure 13: Quadtree child numbering

In previous research work [34], a comparison was done to justify the use of multi-resolution grids in the context of path planning and collision avoidance for semi-autonomous robotic systems operating in complex environments. The schemes under study were evaluated on operations relating to path planning in artificial potential fields.

Following the study of environments of various size and complexity, the multi-resolution representation of the environment was found to be approximately 30% faster in computing the repulsive field and encoding it to the tree structure than its fixed-size equivalent. This improvement is mainly due to the lower number of searches that need to be performed since multi-resolution cells cover large areas of similar occupancy. The multi-resolution structure was also found to improve the free space and attractive field computations as a lower number of cells need to be evaluated, as well as the global path planning method that yields a path defined with fewer steps. Computation times for path planning were also reduced by 30% on average due to fewer steps and neighbor cells that describe the path. We also found that fixed-size cells usually generate a path closer to obstacles and offer much less flexibility in the presence of nearby obstacles than multi-resolution cells.

The approach described above represents the classical method of storing and displaying a multi-resolution occupancy map. However, it presents some deficiencies when searching for neighboring cells, as required for attractive field computation and trajectory planning. In this work, a new method is proposed to lower the memory requirements for storing the grid information while reducing neighbor searching computation times.

3.2.2. Proposed encoding scheme

The goals that the proposed algorithm should fulfill are threefold:

- Provide a compact representation of an environments to lower memory requirements,
- Achieve efficient encoding for rapid computation of neighbor searches to reduce computation times, and
- Remain flexible to allow for online updates to the representation.

To compact the quadtree mapping and provide rapid access to occupancy information, it is proposed to convert the tree structure into a line sequence where the parent levels are placed before their respective infant levels. Therefore, a cell can simply be referenced by its

position in the sequence. To increase clarity, examples presented here are shown for 2D environments, but the techniques have also been adapted for 3D space. A cluttered environment of 16x16 cells is used for the purpose of demonstration, as shown in Figure 14. The grid represents the highest resolution and smallest subdivision where each cell becomes a leaf of the tree structure. Black cells are considered obstacles of unknown occupancy state. A security margin of two cells is shown, computed from the distance table as detailed in section 2.3.1.2, and added around the obstacles and environment boundaries. The margin is shown in Figure 14 as grey cells. The remaining free space, depicted with white cells, becomes apparent with the presence of this margin.

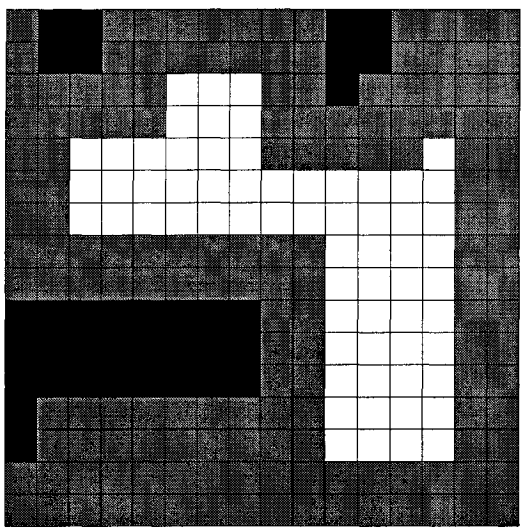


Figure 14: Sample environment with security margin (16x16 grid)

Regions of similar occupancy are regrouped from the free space. The obstacles and cells within the security margin are considered as occupied to ensure safe robot navigation. The environment is then successively divided and analyzed for consistency until the entire workspace has been assigned a consistent value. The cells near the border between the free and occupied spaces are usually in a higher resolution than the rest of the workspace. The multi-resolution free space encoding for the example of Figure 14 is displayed in Figure 15.

One of three occupancy values is assigned to every cell:

- "0" is assigned for a cell that is uniformly occupied, shown in grey,
- "1" is assigned for a cell that is uniformly free, shown in white, and
- "2" is assigned for a cell of non-uniform occupancy: a combination of "0" and "1". This cell is subdivided into 4 quadrants, where the same logic is applied.

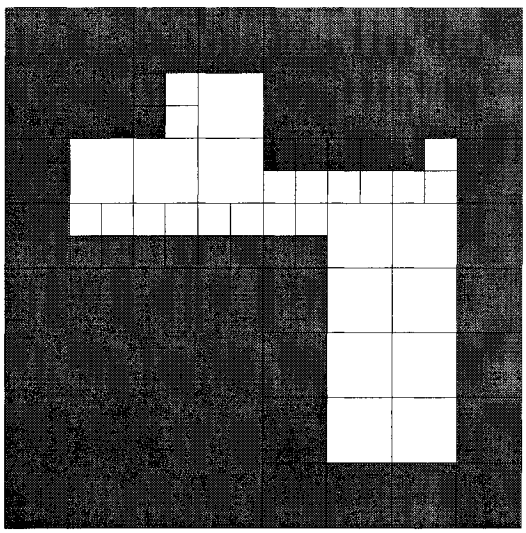


Figure 15: Multi-resolution illustration of the occupancy grid

The tree structure starts at level 0 with the entire modeled area: in this case, it is not of consistent occupancy, therefore, it is of unknown occupancy and assigned "2". For the next level, every non-uniform cell is subdivided into 4 children cells, which are in turn, analyzed for consistency. The process is continued until the smallest resolution is reached. The resulting tree structure that corresponds to the map of Figure 15 is shown in Figure 16. Every children cell is placed in the structure according to the numbering convention proposed in Figure 13.

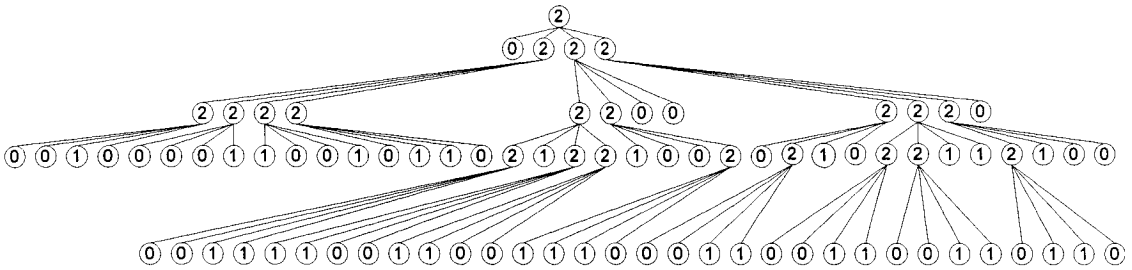


Figure 16: Tree representation of quadtree sequence

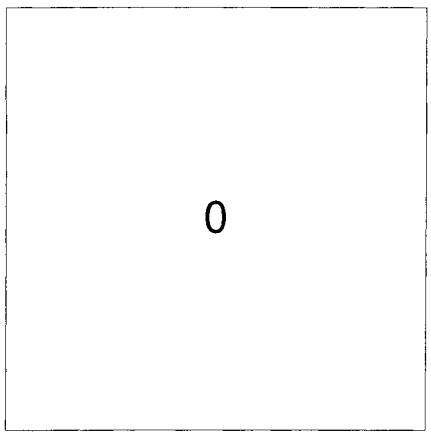
To achieve higher compression, the tree is then converted to a line sequence, starting with the largest level and appending the subsequent levels to the end in the order in which they appear when the tree structure is scanned line by line from the root (the top level). In this example, the resulting sequence is shown in Figure 17:

```
2 0222 2222 2200 2220 0010 0001 1001 0110 2122 1002 0210 2211 2100 0011 1100 1100
1110 0011 0011 0011 0110
```

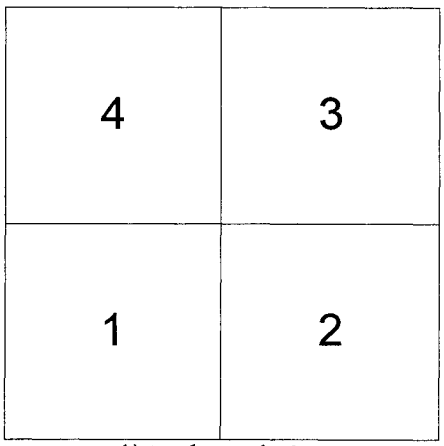
Figure 17: Quadtree sequence

We can now refer to a cell in the occupancy map by its position in the sequence, for example, the first "0" is at position 1, the first "1" is at position 19 and so forth. Upon first inspection, this sequence loses the explicit representation of the position of each cell in the workspace. However, if the ordering of the sequence is known, then the workspace is perfectly mapped into the proposed sequence with much higher compression. By progressively applying the elementary numbering scheme proposed in Figure 13 one level of resolution at a time to the multi-resolution grid of Figure 15, numbers can be associated to each cell which determine their respective position in the compact encoding.

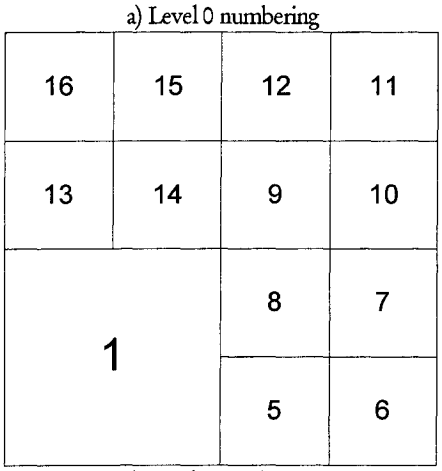
The numbering progression for every level is shown in Figure 18. At level 0, the entire workspace is analyzed for consistency, since the environment is not consistently occupied or empty, a value of "2" is assigned to position 0. The breakdown of the cell position for the following levels is shown in Figure 18a-e: each subdivision follows the same numbering order presented in Figure 13. The numbering scheme represents the position of each cell in the proposed quadtree sequence. Their occupancy status can therefore be directly retrieved from the sequence.



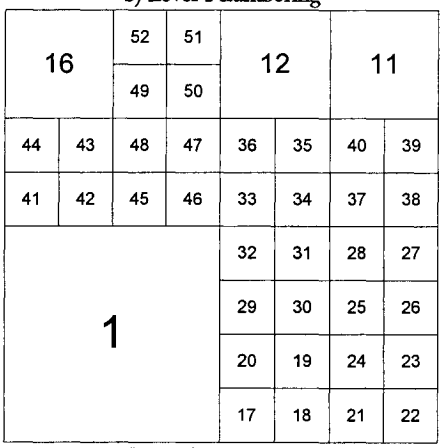
a) Level 0 numbering



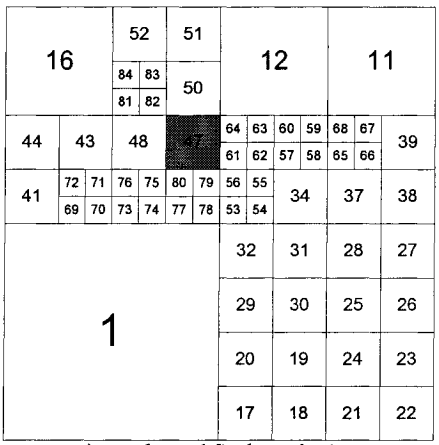
b) Level 1 numbering



c) Level 2 numbering



d) Level 3 numbering



e) Level 4 and final numbering

Figure 18: Environment with quadtree cell numbers

From the encoding sequence, the entire workspace can be rebuilt perfectly even though the compact model contains much fewer elements that its original fixed-size format. The start position for every level can also be stored in an index to reduce computation times when searching through the sequence, in this case [0 1 5 17 53].

In its fixed-size form, the environment above can at best be determined by 16x16=256 cells or 16x16x2=512 bits, assuming 2 bits per cell. With the use of multi-resolution cells, we only need to store the occupancy state of 85 multi-resolution cells, each with only 3 possible values. If using 2 bits to identify the occupancy value, a total of 85*2=170 bits are needed to store the entire environment information. After examining various environments, the average compression ratio of the proposed algorithm is found to be 96% in 2D environments over traditional fixed-size grids, while the compression of 3D environments offered was even more significant.

3.3. Neighbor search

Although multi-resolution grids reduce the number of cells that define the environment compared to classical fixed-size models, the relationship between neighboring cells is more complex. When dealing with those classical occupancy maps based on Cartesian coordinates, border cells can simply be found based on their respective position. However, with multi-resolution grids, these relationships are not as simple. When bringing forth an encoding scheme, this increase in complexity should not offset the advantages of a lower number of cells to evaluate.

Neighbor searching is an important aspect of path planning through potential fields. A poor strategy can significantly reduce the efficiency of the approach. Indeed, during a preliminary implementation of this project, execution times took up to 6 hours with a lousy neighbor finding algorithm. In this section, a method for efficient neighbor finding operating on the encoding scheme described previously is proposed, developed and analyzed with respect to other approaches found in the literature.

3.3.1. *Techniques from the literature*

There are two general approaches of neighbor searching with multi-resolution cells. In the first case, a neighbor is found by searching and analyzing the entire tree structure until the correct neighbor cell is found. This brute force approach is usually computationally expensive even if some researchers have implemented smarter versions of search methods.

Samet [30, 31] proposed such a classical approach to determine the desired neighbor cell. Starting from the current cell and backtracking in the tree, a common parent or grandparent is found. The algorithm then searches the leaf cells of this parent cell until the desired neighbor is found.

The other strategy consists of assigning every cell in the quadtree structure an address based on its position. The neighbor is then found by following a series of predefined computational rules that operate on the addresses. These approaches usually yield better results than the brute-force techniques. Payeur [26] proposed an indexing method to identify individual cells within the tree. A series of algebraic rules is presented to determine the location of a neighbor based on the address of the current cell.

3.3.2. *Proposed approach*

According to the structure of the mapping that has been introduced in section 3.2 which yields small datasets that fully model the environment, a cell is referred to by its position in the sequence. Therefore, its neighbors can also be referred by their position in the same sequence. Starting from this observation, a set of rules is now proposed that will allow to identify neighboring cells from the sequence.

From the position of a cell, a number of properties can be determined about the cell: its location in the environment (in Cartesian coordinates), sibling cells, parent cells and infant cells (based on the knowledge of the tree and the position of the cell). As the cells are organized in a specific order, it is simple to find a neighbor when it is part of the same parent cell (sibling cells), since the neighbor can only be one of 3 candidates in quadtrees or

of 7 candidates in octrees. In this case, the neighbor can be found by adding or subtracting a number based on the position in the parent cell, and therefore in the sequence.

Difficulties occur when the neighbor is not in the same parent cell. The algorithm needs to compare parent cells to make sure that they are sibling cells. This upward movement is done until we are comparing two cells of the same parent. This will happen, at the latest, at the top of the tree. The algorithm, therefore, must compare parent cells until they are sibling cells. In order to avoid the lengthy process of moving up and down the tree sequence as proposed in elementary approaches, the proposed strategy updates the neighbor address at every level.

The neighbor of a cell in a fixed-size tree will always be of the same size. In multi-resolution mapping, the neighbor of a cell can be of a different size compared to the original cell: the same size and level in the sequence, but also larger or smaller. Determining if a neighbor is larger or smaller is more complex and is explained in section 3.3.2.2.

The following sections detail the proposed neighbor finding approach for the case of fixed-size and multi-resolution mappings.

3.3.2.1. Algorithm for fixed-size mapping

In order to provide easier understanding of the proposed approach, an example is shown to explain the concepts used with a fixed-size map, where cells are of identical sizes. The ordering scheme is identical as in Figure 13 in section 3.2.1. Figure 19 shows a 16x16 environment in which the cells have been numbered. Since the previous levels in a fixed-size grid will only contain redundant data to the sequence, only the highest resolution is required and the numbering sequence will commence at the start of this level.

255	254	251	250	239	238	235	234	191	190	187	186	175	174	171	170
252	253	248	249	236	237	232	233	188	189	184	185	172	173	168	169
243	242	247	246	227	226	231	230	179	178	183	182	163	162	167	166
240	241	244	245	224	225	228	229	176	177	180	181	160	161	164	165
207	206	203	202	223	222	219	218	143	142	139	138	159	158	155	154
204	205	200	201	220	221	216	217	140	141	136	137	156	157	152	153
195	194	199	198	211	210	215	214	131	130	135	134	147	146	151	150
192	193	196	197	208	209	212	213	128	129	132	133	144	145	148	149
63	62	59	58	47	46	43	42	127	126	123	122	111	110	107	106
60	61	56	57	44	45	40	41	124	125	120	121	108	109	104	105
51	50	55	54	35	34	39	38	115	114	119	118	99	98	103	102
48	49	52	53	32	33	36	37	112	113	116	117	96	97	100	101
15	14	11	10	31	30	27	26	79	78	75	74	95	94	91	90
12	13	8	9	28	29	24	25	76	77	72	73	92	93	88	89
3	2	7	6	19	18	23	22	67	66	71	70	83	82	87	86
0	1	4	5	16	17	20	21	64	65	68	69	80	81	84	85

Figure 19: Fixed resolution grid with cell numbering

In general, the neighbor finding rules depend only on the position of the cell to inspect and the direction we wish to explore: the northern neighbor for example. Observing how addresses are modified when a movement occurs in each of the possible directions allows us to define generic rules that can be applied repetitively over the entire map. These rules are encoded as a lookup table for efficiency. Following these rules, the neighbor location can be updated according to the start position and the information found in the lookup table, shown in Table 1. There are two inputs to each table:

- The position of the current cell in its parent: it is represented by the center number in each table, underlined, and
- The direction of the neighbor to find: in 2D this can be one of eight possibilities, as shown in Figure 20.

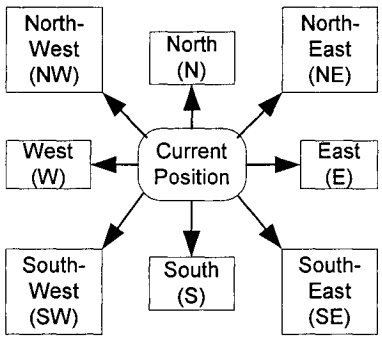


Figure 20: Neighbor directions

For each direction, two outputs are found in the table: the adjustment required to reach the neighbor and the direction of the parent neighbor. The adjustment represents the location of the neighbor in its parent at the current level. These rules are applied recursively until there is no direction returned, which means that the two cells are part of the same parent.

W+2	+3	+2
W+1	0	+1
SW+2	S+3	S+2

Table 1a

+2	+1	E+2
-1	1	E-1
S+2	S+1	SE+2

Table 1b

N-2	N-1	NE-2
+1	2	E+1
-2	-1	E-2

Table 1c

NW-2	N-3	N-2
W-1	3	-1
W-2	-3	-2

Table 1d

Table 1: 2D neighbor searching rules

For a 2D case, the main steps of the proposed neighbor finding approach can be summarized as follows:

1) Set $neighbor_{position}$ and $parent_{position}$ to be equal to the position in the sequence of the cell currently analyzed and iteration i to 0.

2) Determine the position of the current cell in its parent (0, 1, 2 or 3): found by taking the remainder of the position in the sequence of the cell analyzed divided by 4.

3) Look up adjustment value and parent direction in Table 1 based on the remainder computed in step 2).

4) Adjust position according to the position of the parents: This position is updated to be: "$neighbor_{position}[t+1]=neighbor_{position}[t]+offset*4^{i}$ ", where offset represents the neighbor adjustment value found in 3) and "i" represents the iteration.

5) If there is a parent neighbor direction provided by Table 1, it means that the starting cell and its neighbor are in different parent cells, then

   a. Regroup all children to form the parent cell of the current cell by taking the integer value of the current cell position divided by 4, to update the parent cell's address: $parent_{position}$ now becomes $parent_{position}/4$.

   b. Increment iteration number, i.

6) Repeat 2-5 until there is no parent neighbor direction. Then, the final $neighbor_{position}$ value gives the position of the desired neighbor cell in the sequence.

To clearly illustrate the procedure, an example is detailed based on Figure 19 where the cell analyzed is at position 111, from which we wish to determine the NW neighbor.

1) We set $neighbor_{position} = parent_{position} = 111$, and iteration $i=0$.

2) remainder($parent_{position}/4$)=remainder(111/4)=3, therefore, we look at Table 1.d.

3) In the NW direction, we find NW-2. Since a direction (NW) is returned, the two cells are not in the same parent, so we must iterate again.

4) We update $neighbor_{position}$ =111-2*4^$iteration$ =111-2*4^0 =109.

5) The new position to examine, $parent_{position}$, is updated to be the parent of the starting cell in the NW direction that is defined as the integer part of ($parent_{position}/4$)=(111/4)=27, and the iteration is incremented to 1.

6) The second iteration brings: remainder($parent_{position}/4$)=remainder(27/4)=3, therefore, we look at Table 1.d.

7) In the NW direction, we find NW-2 again. Since a direction (NW) is returned, the two cells are not in the same parent, so we must iterate again.

8) We update $neighbor_{position} = neighbor_{position} - 2*4\hat{}increment = 109 - 2*4\hat{}1 = 101$.

9) The new position to examine, $parent_{position}$, is updated to be the parent of the starting cell in the NW direction that is defined as the integer part of $(parent_{position}/4) = (27/4) = 6$, and the iteration is incremented to 2.

10) The third iteration gives: remainder$(parent_{position}/4)$ =remainder$(6/4)$ =2, therefore, we look at Table 1.c.

11) In the NW, we find N-2. Since a direction (N) is returned, the two cells are not in the same parent, so we must iterate again.

12) We update $neighbor_{position} = neighbor_{position} - 2*4\hat{}increment = 101 - 2*4\hat{}2 = 69$.

13) The new position to examine, $parent_{position}$, is updated to be the parent of the starting cell in the N direction that is defined as the integer part of $(parent_{position}/4) = (6/4) = 1$, and the iteration is incremented to 3.

14) The fourth iteration gives: remainder$(parent_{position}/4)$ =remainder$(1/4)$ =1, therefore we look at Table 1.b.

15) In the N, we find +1, there are no other directions, we are therefore comparing two cells of the same parent.

16) We update $neighbor_{position} = 69 + 1*4\hat{}3 = 133$.

Therefore, we verify that the NW neighbor of 111 is 133. The advantage of this approach is that the neighbor position is updated at each iteration without the need to search in through the upper levels of the sequence to reach the desired neighbor cell. Hence, there is no need for back tracking in the tree to determine its position, which saves a notable amount of computation and access to the model. This method is extended to multi-resolution cells in the following section.

3.3.2.2. Algorithm for multi-resolution mapping

This approach works well for fixed-size cells, but the requirements are different with multi-resolution cells. In the latter case, the method is similar except that the entire tree structure must be taken into consideration because one might be looking for neighboring cells of different sizes. With multi-resolution grids, there are three different cases that can happen:

1) Finding a neighbor of the same size
2) Finding a neighbor of smaller size
3) Finding a neighbor of larger size

To overcome this situation, the neighbor search algorithm is applied to an equivalent sequence of fixed-size cells, similar to Figure 19 that needs to be computed from the multi-resolution representation. The search for neighbors is executed as described in the previous section on the intermediate fixed-size map. The overhead is then the necessary conversion of the cell examined between the fixed-size and the multi-resolution maps. The flow diagram of the modified method follows these steps:

1) Convert the multi-resolution cell number position to its fixed-size equivalent position. Only the equivalent cell number is needed, not the entire grid. This is done by scanning the sequence for each of the previous levels, counting the number of uniform cells and adjusting the fixed-size cell's location according to the following rules.

    a. Set $current_{Position}$ to be equal to the current cell position in the multi-resolution sequence and *iteration* to 0.

    b. Determine the level where $current_{Position}$ is found and the smallest position value belonging to that level, arr_{Level} and $level_{Start}$. The level start index can be used to reduce computation. Initialize $fix_{cell} = current_{Position} - level_{Start}$.

c. Determine the position of the parent cell in the preceding level: $parent_{Pos}$=integer($(current_{Position} - level_{Start})/4$)+1. This represents the number of non-uniform cells in the previous level that have been subdivided before the current parent. Increment *iteration* by one, decrement cur_{Level} by one.

d. Update $level_{Start}$ to become the smallest position of cur_{Level},

e. Include the presence of uniform cells to the fixed-size equivalent mapping by counting the number of uniform cells, *num01*, that come before the current position and adjusting for the level. Also, $num2 = parent_{Pos} + num01$.

f. Update fix_{cell} to consider the effects of uniform cells: $fix_{cell}[t+1] = fix_{cell}[t] + num01 * 4^{iteration}$. This updated number represents the position of the current cell in the fixed-size sequence once the uniform cells are considered in the current level.

g. Repeat c) to f) until level 1 is reached, update $current_{Position} = num2 + level_{Start}$.

2) Execute the same neighbor finding algorithm than with fixed-size.

3) Convert the fixed-size neighbor to its multi-resolution equivalent by checking the higher levels of the tree until a uniform cell (entirely free or entirely occupied) is found at the neighbor position and adjusting based on the number of uniform cells found, or until the current level is reached. If at this level a non-uniform cell is returned, neighboring children must be found. This is done by following this procedure:

a. Determine level of original cell (which can be stored from 1b), denoted cur_{Level}.

b. Set $multires_{neighbor} = fix_{cell}$ and $levelExamined$=1.

c. At level *levelExamined*, determine the position of $multires_{neighbor}$ in the multi-resolution sequence: $curLevFix = multires_{neighbor} / 4^{(curLevel - levelExamined)}$, only taking the integer of the division.

d. Determine the smallest position value belonging to *levelExamined*, denoted $positionLevel_{Start}$.

e. If the occupancy of position $positionLevel_{Start}$ + *currLevFix* in the multi-resolution sequence is found to be uniform, then:

   i. A neighbor of larger or equal size is found at level *levelExamined*, the algorithm stops and the neighbor position returned is $positionLevel_{Start}$ + *currLevFix*.

   ii. Otherwise, if the algorithm has reached the original level, then:

      A. The neighbor position is not uniform at the original level, therefore, stop this routine and determine the children of smaller sizes by following the routine listed in Figure 21.

      B. Otherwise, go to f).

f. Remove the effects of uniform cell in the fixed-size sequence by counting the number of uniform cells between $positionLevel_{Start}$ and $positionLevel_{Start}$ + *currLevFix*, denoted *num01*.

g. Update $multires_{neighbor}$ to be the new position considering the effects of uniform cells in the sequence: $multires_{neighbor} - num01 * 4^{(currLevel - levelExamined)}$. Increment *levelExamined* by one and repeat from c).

Three types of neighbors can be returned from the preceding sequence: a neighbor of the same size is found at e) when $positionLevel_{Start}$ + *currlevFix* at *levelExamined* = $curr_{Level}$. Secondly, a larger neighbor is found when $positionLevel_{Start}$ + *currlevfix* at *levelExamined* < $curr_{Level}$. Finally, a set of rules has been established as shown in Figure 21 to determine the location of the neighboring sub-cells when a non-uniform neighbor is found. Once such a neighbor is returned, its children cells are needed in order to accurately represent the neighbor. In Figure 21, the location of the appropriate children cells are listed based on their position in

the parent and the direction of the neighbor search. This approach can be called recursively to find a sub-neighbor smaller than one level.

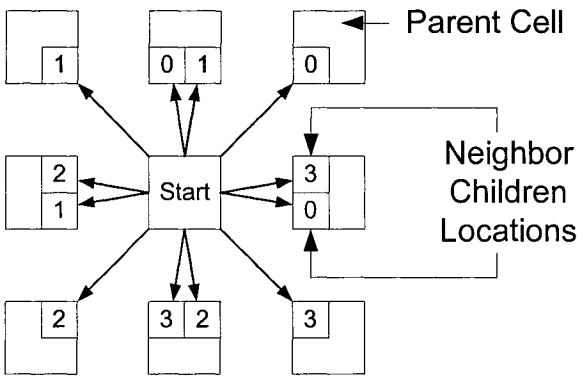


Figure 21: Sub-neighbor listing

For clarity, a generic example is detailed. From cell 47 in Figure 18, we want to find I) the north-eastern (larger size), II) the eastern (smaller size) and III) the northern (same size) neighbors. Following the algorithm presented previously, we have:

1) Convert 47 to its equivalent fixed-size cell position:

   a. Set $current_{position}$=47 and $iteration$=0.

   b. From the sequence and the start position index of [0 1 5 17 53], we find that 47 is located in cur_{Level} =3, and level 3 starts at $level_{Start}$ =17. We initialize fix_{cell} = $current_{position}$ - $level_{Start}$ =47-17=30.

   c. The parent of fix_{cell} represents the number of non-uniform cells in the preceding level denoted $parent_{Pos}$=integer(($current_{position}$ - $level_{Start}$)/4)+1=integer(30/4)+1=8. We update $iteration$=$iteration$+1=0+1=1 and cur_{Level} = cur_{Level} -1=3-1=2.

   d. For level 2, the start position index lists $level_{Start}$ =5.

   e. In order to find the position of the parent, we must count the number of uniform cells prior to the parent position. When the 8th "2" is located, 2 such cells have been counted (at positions 11 and 12 in the sequence, as

shown in Figure 17), we set $num01=2$ and $num2= parent_{Pos} + num01=8+2=10$.

f. We update the fixed-cell value to include uniform cells in the sequence: $fix_{cell}[t+1]=fix_{cell}[t]+num01*4^{iteration}=30+2*4^1=38$.

g. We find that cur_{Level} is not 1, we repeat the process with $current_{position} =num2+ level_{Start} =10+5$.

h. Repeating c), we find that $parent_{Pos}$=integer(($current_{position}$ - $level_{Start}$)/4)+1=(15-5)/4+1=3, which represents the number of non-uniform cells in the preceding level. We update $iteration=iteration+1=1+1$ and $cur_{Level} = cur_{Level} - 1=2-1=1$.

i. For level 1, the start position index lists $level_{Start} =1$.

j. In order to find the position of the parent, we must count the number of uniform cells prior to the parent position. When the 3rd "2" is located, 1 such cell has been counted (at position 1), we set $num01=1$ and $num2= parent_{Pos} + num01=3+1=4$.

k. We update the fixed-cell value to include uniform cells in the sequence: $fix_{cell}[t+1]=fix_{cell}[t]+num01*4^{iteration}=38+1*4^2=54$.

l. We find that $cur_{Level} =1$, therefore we can stop; we have found that the fixed-size equivalent position number to 47 is 54.

2) Using the method described in section 3.3.2.1, we find the following fixed-cell neighbors, as shown in Figure 22.

I) The north-eastern neighbor to 54 is 44

II) The eastern neighbor to 54 is 35

III) The northern neighbor to 54 is 57

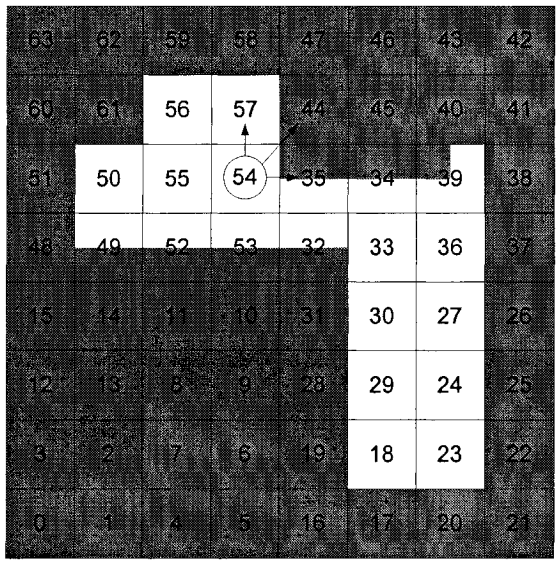


Figure 22: Fixed-size representation of example

3) Converting back to multi-resolution:

I) 44 represents the north-eastern neighbor in its fixed-size equivalent. Following the method described above, we need to identify its equivalent multi-resolution position, which is obtained as follows:

a. We start are still at $curr_{Level}$ =3.

b. We start with $multires_{neighbor}$ =44 and $levelExamined$ =1.

c. The parent of $multires_{neighbor}$ in level 1 is also its grandparent located two levels above. We find its position in the tree with $currLevFix = multires_{neighbor} / 4^{currLevel - levelExamined} = 44/4^{3-1} = 44/16 = 2$ (taking the integer value).

d. From the sequence and the start position index of [0 1 5 17 53], we find that level 1 starts at position $positionLevel_{Start}$ =1.

e. In the sequence, we determine that $(currLevFix + positionLevel_{Start}) = 2+1 = 3$ is not uniform: it has value of "2".

B. We also find that $curr_{Level}$ is not $levelExamined$. Therefore, we must continue.

f. In the sequence, between position $positionLevel_{Start}$ =1 and $positionLevel_{Start}$ + $currLevFix$ =3, we count 1 uniform cell (at position 1), therefore, $num01$=1.

g. We eliminate the effects of the large uniform cell by $multires_{neighbor}$ = $multires_{neighbor}$ $-num01*4^{currLevel-levelExamined}$=44-1*$4^{3-1}$=44-16=28. We increment $levelExamined = levelExamined$ +1=1+1=2.

h. Repeating c), we find $currLevFix$ =$multires_{neighbor}$ /$4^{currLevel - levelExamined}$=28/$4^{3-2}$=28/4=7.

i. From the sequence and the start position index of [0 1 5 17 53], we find that level 2 starts at position $positionLevel_{Start}$ =5.

j. In the sequence, we determine that the position ($currLevFix$ + $positionLevel_{Start}$) =7+5=12 is uniform: it has value of "0",

   A. We can stop and confirm that the multi-resolution neighbor is located in position 12.

II) 35 represents the eastern neighbor in the fixed-size mapping, converting it using the similar approach as above, we find that the neighbor (36) at the current level (level 3) is not uniform; therefore, using the proposed approach shown in Figure 21, the northern eastern neighbor will consist of smaller cells. 36 represents the 3rd "2" in level 3, so the eastern neighbors will be the western cells of the 3rd series of cells at the children level of 4, these are 61 and 64 which are both uniform.

III) In similar fashion, we find that the multi-resolution cell that corresponds to the northern neighbor is 50 and is at the same level than the original cell, 47.

The proposed method aims to provide an efficient neighbor-searching algorithm while keeping memory requirements low. Only the environment data in its sequence format is stored. As opposed to other methods found in the literature, there is no need for any other data to be stored, such as addresses or tree information.

In a dynamic environment, the tree structure can easily be updated, as new data is made available. Depending on the new information acquired, the sequence will either expand or contract, therefore, the positions in the sequence will undergo changes that have to be taken into account. Since the sequence can be updated without difficulty and the neighbor search method only depends on the current cell position, the proposed sequence can be well suited to a dynamic environment.

Since the cell number represents the position of the cell in the sequence pointer, occupancy information is also easily accessible: once the position of the cell is found, its occupancy is also found at the same position in the sequence: for instance, the occupancy of cell 47 is located in position 47 of the quadtree sequence. This quick access to information increases the efficiency of the algorithm and reduces computation times in building the attractive field and neighbor searching where occupancy information is important.

Similar rules have been developed for neighbor searching in 3D. For space consideration, the details are reported in Appendix A.

3.4. Comparison and results

The proposed method offers an effective approach in storing environment data. Computation times for neighbor searches are lowered in comparison with standard grids, while storing the environment data at a fraction of the cost. The proposed method offers quick and accurate neighbor searches without the need for extravagant searches throughout the quadtree data. In order to evaluate the performance of the proposed approach, a comparison using test cases of different sizes is done with the backtracking approach presented by Samet [30, 31] and an addressing scheme proposed by Payeur [26]. Both approaches use the same traditional quadtree format. For these test cases, all three approaches are applied to build a multi-resolution representation and to determine the neighbor positions for all cells in the free space in similar fashion to attractive field computation. In order to provide more accurate results, the entire test cases are repeated

and the computation times are noted after all the iterations. Table 2 displays computation times (in ms) and the size of the encoding (in number of bits) for the respective test cases.

Test case number	Iteration	Backtracking quadtree		Addressing quadtree		Sequence	
		Computation (ms)	Size (bits)	Computation (ms)	Size (bits)	Computation (ms)	Size (bits)
1	1600	139 060	336	16 043	336	16 010	168
2	1600	141 490	1 280	18 620	1 280	17 680	640
3	20	38 550	7 744	3 920	7 744	4 570	3 872
4	20	51 040	8 512	4 340	8 512	4 680	4 256
5	20	111 550	15 648	22 980	15 648	21 900	7 824
6	20	15 910	5 472	2 900	5 472	2 930	2 736
7	20	21 010	5 520	2 800	5 520	2 970	2 760
8	20	13 740	4 992	2 600	4 992	2 470	2 496
9	20	17 690	5 344	2 740	5 344	2 940	2 672
10	20	16 810	5 216	2 640	5 216	2 650	2 608

Table 2: Comparison results for multi-resolution encoding and neighbor search algorithms

In Table 3, the environments used for the test cases are displayed; their respective heights and widths are also listed below each figure and adjusted in size for ease of viewing. They were chosen to represents heavily cluttered environments, and similar environments to the test case scenarios of Chapter 5. The fifth environment represents obstacle points randomly spread over the workspace was chosen in order to generate a highly complex tree model with many cells.

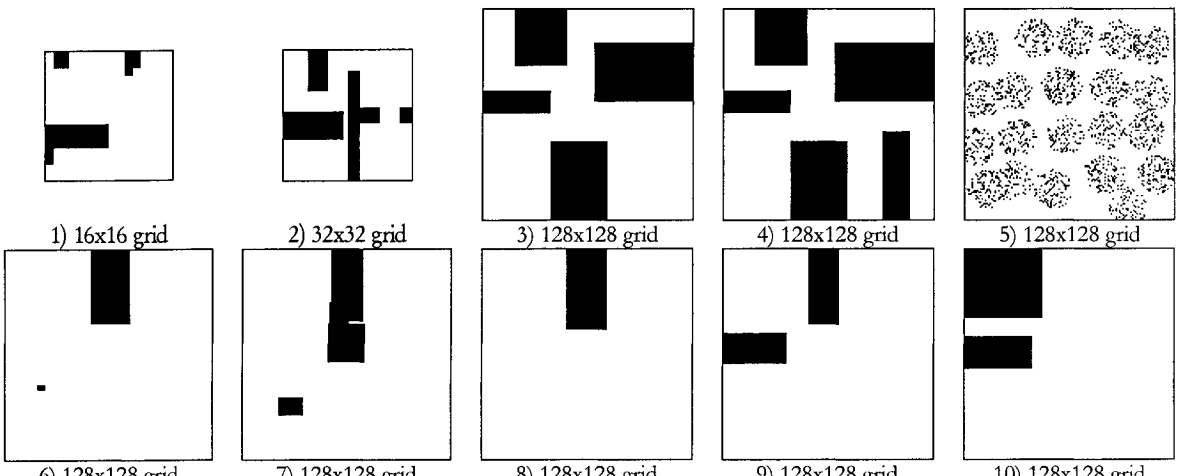


Table 3: Test case environments

The approach developed by Samet [30, 31] has been tested against the proposed approach. As expected, the proposed algorithm offered an average reduction in

computation times of 85%. The approach proposed by Samet uses too many searches in the tree data. Also, a direct relationship, like neighbor rules, is not used to validate whether a cell is a neighbor or not. Both of these factors slow down neighbor finding routines considerably. Since the approach brought forth by Samet uses the typical tree representation, the proposed approach requires half the memory to model the workspace. The important reduction in computation using the proposed method offers a significant advantage over the approach presented by Samet.

A comparison between the proposed algorithm and that of Payeur [26] was also performed to validate the proposed strategy. The two approaches are similar, but differ in the way cells are identified. Computation times were nearly identical for the two methods: a 2.5% improvement for [26]. While the addressing scheme used in [26] is easier to visualize, it does not provide as quick of an access to the cell occupancy status as obtained with the sequence encoding. In the proposed approach, the neighbor location also represents the precise location in memory of its occupancy status.

The proposed method does not add any more memory requirements than that of the tree sequence itself, in contrast with [26] which uses the traditional tree representation and other approaches where the tree structure and pointer information must be stored. Since both approaches provide similar computation times, we find that the proposed method is slightly better since it requires half the memory.

3.5. Conclusion

The method proposed in this chapter offers a simple and efficient approach to encode and analyze an environment. It has shown similar complexity to other algorithms found in the literature while reducing memory requirements. From the results obtained, we conclude that it is well suited for the attractive field computation and global path planning phase that will be discussed in the next chapter.

CHAPTER 4: PATH PLANNING AND ROBOT CONTROL

4.1. Introduction

In order for the robot to be able to complete the task it has been assigned to do, it is essential to have it move safely across the workspace. The goal of the path planning method is to determine a sequence of configurations for the robot to move around obstacles and avoid collisions while reaching a desired goal. This work follows an important trend in manipulator path planning which consists of finding a safe trajectory in two steps:

1. A planning phase where the path is determined for the end effector only using the multi-resolution occupancy model, which is usually done offline, and
2. An online tracking phase to follow from the initial path, in which a sequence of robot arm configurations is determined to satisfy the path of the end effector while avoiding contact with the obstacles for the entire structure. The robot moves towards the updated position while the controller finds the next position.

The first path found corresponds to an approximation of the trajectory of the end effector and is based on a "global path planning" method. It is generally preferred for this path to be at conservative distance to obstacles. This trajectory is found by following the down slope of the discrete attractive potential field, as detailed in Chapter 2. This approach uses the advantages of path planning algorithms for mobile robots, that is, in both cases; only a fixed point or structure is considered for movement.

For the "path tracking" phase, that extends the trajectory definition to the whole structure, the rest of the robot arm is configured to follow the path of the end effector. Since the direct computation of manipulator parameters through inverse kinematics is a difficult and inflexible process that varies for each robot architecture, an alternative method is proposed to combine the model of the robot arm with the local environment mapping and output the required sequence of joint values. In this chapter, an adapted fuzzy logic

controller that considers the configuration of the robot and environment obstacles is proposed for its simplicity and robustness to different robotic arms.

4.2. Global path planning

4.2.1. Neighbor search and movement of end effector

The purpose of this step is to provide the local robot controller with a generic path to follow during the more refined path tracking phase. We wish to generate a guideline path for the end effector to travel. This path allows leeway to reduce computation times and the occurrence where the structure of the manipulator may lead to unreachable positions along this initial path. The global path serves as a general indication and is obtained with methods very similar to those found for path planning in mobile robotics.

The discrete attractive field computed from the quadtree or octree mapped as a sequence, as proposed in Chapter 3, is the main guide in finding this path. From the starting position and configuration of the robot, the algorithm follows the down slope of the attractive field until the goal or target position is reached. However, as this is a method mainly directed towards mobile robot, or the movement of a single entity, the extension to the control of a manipulator implies that the following assumptions are made:

- Given the current position of the end effector, there exists a configurable position of the robot, i.e. the position must be physically reachable by the arm without any collisions.
- The target position is in free space and is reachable with the end effector, i.e, it is not inside a closed region.

As the result of this first planning phase is considered a coarse path, one that gives a general indication of the movement of the end effector, the output path does not need to be entirely in its highest resolution. Therefore, this global approach can be computed using the multi-resolution cells that have been presented in Chapter 3 to minimize the number of neighbor searches and improve performance. This path is consequently a sequence of multi-

resolution cells from the starting cell position to the goal cell position used to guide the end effector in the environment. An example of a 2D environment is shown in Figure 23 with a planned coarse path. The global path of the end effector is defined by the sequence of 7 multi-resolution cells shown in grey, starting with "S" and reaching the goal denoted "T". To minimize the occurrence of a collision, this general path suggests that the end effector successively reaches the center of these cells and the joint border with the next, as indicated with the small squares in Figure 23. The path is to be divided into smaller steps in the path tracking phase.

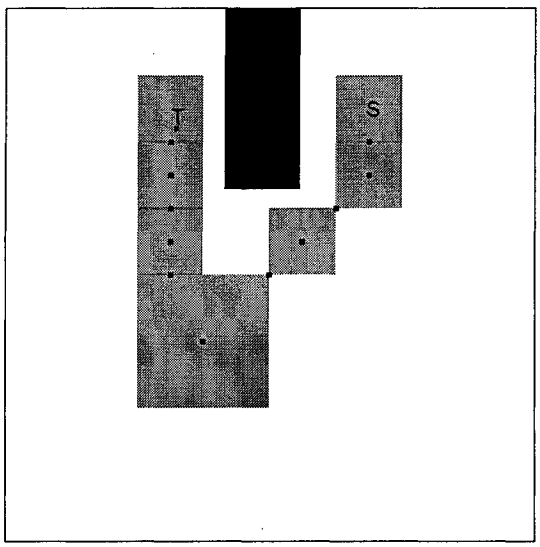


Figure 23: End effector path

4.3. Path tracking

4.3.1. Complexities of analytical solution

Manipulator arms are inherently more difficult to analyze than mobile robots because when a part of the structure moves, it can affect the rest of the structure, creating difficult situations that do not appear in mobile robot path planning.

The relation between the robot parameters and the end effector pose[1] is a model called the forward kinematics (FK). The forward kinematics is relatively easy to compute as each

[1] The pose of an object is defined as its position and orientation with respect to a reference frame.

joint is assigned a transformation with respect to the previous one. Regrouping these transformations will yield the forward kinematic expression. A common method to find this is proposed by Denavit and Hartenberg. For each joint, the matrix displayed in Equation 3 is found.

$$A_i = \begin{pmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & L_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & L_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (3)$$

Where θ_i represents the joint angle value and the other symbols are internal robot parameter, such as joint lengths (l_i, d_i) and angle offset (α_i). When a robot consists of two or more joints, similar matrices are multiplied together. This yields a large expression even for a relatively simple robot. The ensuing multiplication represents the forward kinematics of the robot and defines the pose of the effector with respect to its fixed base as a function of the robot internal parameters and joint values.

While this relation provides a lot of information for the system, it leads to a computationally expensive process for robot control. Usually a desired pose to reach is given and the robot parameters (such as joint angles) need to be calculated to reach that pose, this process is called the inverse kinematics (IK). Finding the inverse kinematics (IK) of a robot has always been a difficult endeavour. The complexity grows significantly as a manipulator includes more degrees of freedom (DOF).

The inverse kinematics is found by determining relationships between the joint angles and the pose parameters. Finding the precise IK equations is very complex and often leads to very complicated or even undefined closed-form solutions. When working with slightly redundant manipulators, finding an accurate solution is often a daunting task. The redundancy of the robot arm can also introduce many valid solutions that must be sorted and evaluated. In order to bypass the expensive computation of IK equation and the brute

force approach of some heuristic approaches, an original adaptation of a fuzzy logic controller is proposed to exploit the advantages of fuzzy logic.

4.3.2. *Fuzzy logic based path tracking*

Fuzzy logic systems have existed for some time [38], in order to find a way to simplify the control of many systems. According to [40], fuzzy logic brings five interesting features to a system:

- A robust controller that does not require precision or noise-free inputs: this is important as sensors don't offer complete precision. The resulting output follows a smooth function even though there may be numerous inputs.
- The controller depends on user-defined functions and parameters: these can be easily changed to tweak the performance or to add new inputs.
- Any sensors that provide knowledge on the state of the system can be used as an input. It allows the system to stay low-cost and low-complexity.
- Because of a rule-based operation (IF-THEN-ELSE), any number of inputs can generate any number of outputs.
- Fuzzy logic can control non-linear problems that would be very difficult or impossible to solve using traditional or analytical methods.

The last item is very interesting to manipulator arms for which the inverse kinematics is usually difficult or impossible to solve. While some papers such as [22] use fuzzy logic to control manipulator arms, the literature does not propose an effective approach based on fuzzy logic for path planning of a manipulator in a cluttered environment. This chapter presents an exploration of such a solution inspired by techniques that have been developed for empty working environments.

The proposed approach uses a standard fuzzy logic controller scheme which is made up of 3 steps:

1) Fuzzification

2) Rule evaluation

3) Defuzzification

Appendix B summarizes these three steps for a generic fuzzy logic controller.

4.3.2.1. Fuzzy logic applied to manipulator arms

The direct or analytical computation of the inverse kinematics is a difficult task especially when dealing with redundant robots. In order to reduce these constraints, commercial robots are usually designed to simplify the inverse kinematics equations. While this is an ingenious idea, it often limits the flexibility of the robot. The approach proposed in this work aims at determining a suitable series of configurations for any robot to move in an arbitrary environment. A fuzzy logic approach should enable us to determine robot configurations that match the pre-defined end effector's path while minimizing the joint movements and avoiding collisions.

The proposed system pursues the following objectives:

- To determine joint configurations that minimize the changes required to reach a given position, that is to achieve small changes between consecutive positions and create a smooth trajectory.
- To separate the general path found in the global path planning phase into smaller, intermediate steps so that collisions can be avoided while joint movements are kept relatively small, therefore reducing the occurrence of singular movements.
- To have an algorithm that can easily include repulsive forces exerted by nearby obstacles.
- Keep internal parameters constant or auto-adaptable when changing environment or robot architecture.

Numerous papers regarding fuzzy logic such as [3], [10] and [35] have been published in the literature. But, as explained in section 2.5.1, the literature does not yet provide a robust solution to manipulator path planning. In [22], Nedungadi proposed an approach to control manipulator movements using fuzzy logic. However, the author only considers a planar robot and the approach does not consider a cluttered workspace, as the goal of the robot is to follow a pre-determined path in a free environment. These introduce major limitations but the concepts can be extended to 3D manipulators working in a cluttered environment. The approach proposed here addresses these aspects and introduces the concept of repulsive forces to the fuzzy logic system presented by Nedungadi to handle collision avoidance in generic 2D and 3D environments.

4.3.2.2. Overview of Nedungadi's approach

The goal of the fuzzy controller can be summarized as to "determine the joint parameters of the robot that satisfy a given position of the end effector to be reached". The algorithm seeks to modify the joint angle values based on the target position considering the current state of the system. Instead of using absolute values for the position, the algorithm only considers the required displacement: how far does the end effector have to move to reach the position.

The state of the robot is also characterized by the rate of movement of the end effector for a unit change in joint angle, which is called the end effector velocity, or the partial derivatives of the forward kinematics equations in velocity. Hence, instead of solving the inverse kinematics, this approach only needs the forward kinematics representation and its first derivative, which is straightforward to compute.

The angle change required on a joint is a function of the end effector displacement needed and the rate of change of the joint (in distance per unit angle). Nedungadi proposes Table 4 to be used as a rule base or fuzzy associative memory (FAM). In this table, input 1 represents the position displacement of the end effector that needs to be obtained while input 2 represents the velocity of the end effector with respect to the current joint or the

end effector displacement for a positive unit change in joint angle. Both of these values are in their fuzzy state and are classified into 7 different values: negative large (NL), negative medium (NM), negative small (NS), zero (Z), positive small (PS), positive medium (PM) and positive large (PL). The output follows the same logic: by matching the two inputs, one of 7 different fuzzy values is found.

		Input 1=effector displacement required						
Input 2=Velocity of effector for positive joint change		**NL**	**NM**	**NS**	**Z**	**PS**	**PM**	**PL**
	NL	PL	PM	PS	Z	NS	NM	NL
	NM	PL	PM	PS	Z	NS	NM	NL
	NS	PL	PM	PS	Z	NS	NM	NL
	Z	Z	Z	Z	Z	Z	Z	Z
	PS	NL	NM	NS	Z	PS	PM	PL
	PM	NL	NM	NS	Z	PS	PM	PL
	PL	NL	NM	NS	Z	PS	PM	PL

Table 4: Fuzzy association bank

The change in joint angle will be large if the displacement required is large. The same deductions can be made for small displacements. End effector velocity, however, only affects the sign of the angle change. For example, if the end effector velocity is negative for a positive change in joint angle (input 2), then the corresponding angle adjustment must be negative (output) to reach a positive position displacement (input 1).

The process is applied successively for every joint. If after the first iteration the target position has not been reached, the entire procedure is repeated with the updated robot configuration until the target position is reached. Figure 24 illustrates the steps used.

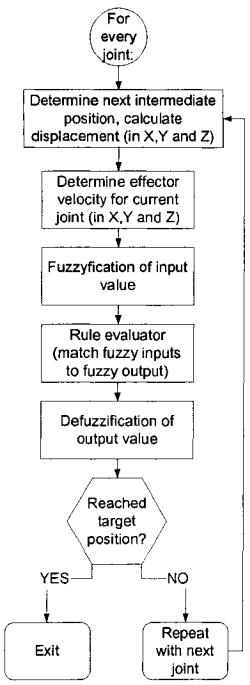


Figure 24: Fuzzy logic block diagram (without collision detection)

As the first input represents the displacement required for the end effector; it is encoded as *dx* and *dy* in Cartesian coordinates. In 3D workspace, a third displacement along the Z axis is added and computed similarly. For the second input, determining the end effector velocity is usually a simple process as the forward kinematics is a combination of sines and cosines of angles. For example, using a planar robot with 4 degrees of freedom (DOF), shown in Figure 25, we find the forward kinematic relationships displayed in Equation 4.

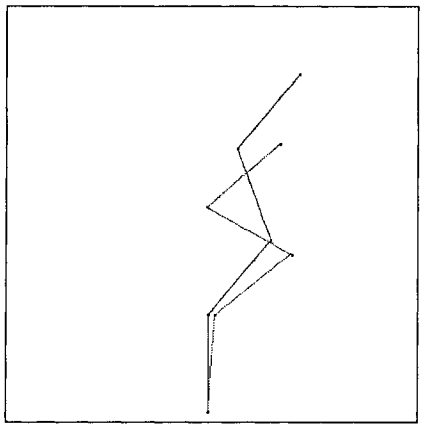


Figure 25: Sample 4-DOF robot at two positions

$$x = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) + L_3 \cos(\theta_1 + \theta_2 + \theta_3) + L_4 \cos(\theta_1 + \theta_2 + \theta_3 + \theta_4)$$
$$y = L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2) + L_3 \sin(\theta_1 + \theta_2 + \theta_3) + L_4 \sin(\theta_1 + \theta_2 + \theta_3 + \theta_4) \quad (4)$$

The X and Y components of the end effector velocities for each joint are given by Equation 5.

$$\frac{\partial x}{\partial \theta_1} = V_{X1} = -(L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2) + L_3 \sin(\theta_1 + \theta_2 + \theta_3) + L_4 \sin(\theta_1 + \theta_2 + \theta_3 + \theta_4))$$
$$\frac{\partial x}{\partial \theta_2} = V_{X2} = -(L_2 \sin(\theta_1 + \theta_2) + L_3 \sin(\theta_1 + \theta_2 + \theta_3) + L_4 \sin(\theta_1 + \theta_2 + \theta_3 + \theta_4))$$
$$\frac{\partial x}{\partial \theta_3} = V_{X3} = -(L_3 \sin(\theta_1 + \theta_2 + \theta_3) + L_4 \sin(\theta_1 + \theta_2 + \theta_3 + \theta_4))$$
$$\frac{\partial x}{\partial \theta_4} = V_{X4} = -(L_4 \sin(\theta_1 + \theta_2 + \theta_3 + \theta_4))$$
$$\frac{\partial y}{\partial \theta_1} = V_{Y1} = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) + L_3 \cos(\theta_1 + \theta_2 + \theta_3) + L_4 \cos(\theta_1 + \theta_2 + \theta_3 + \theta_4) \quad (5)$$
$$\frac{\partial y}{\partial \theta_2} = V_{Y2} = L_2 \cos(\theta_1 + \theta_2) + L_3 \cos(\theta_1 + \theta_2 + \theta_3) + L_4 \cos(\theta_1 + \theta_2 + \theta_3 + \theta_4)$$
$$\frac{\partial y}{\partial \theta_3} = V_{Y3} = L_3 \cos(\theta_1 + \theta_2 + \theta_3) + L_4 \cos(\theta_1 + \theta_2 + \theta_3 + \theta_4)$$
$$\frac{\partial y}{\partial \theta_4} = V_{Y4} = L_4 \cos(\theta_1 + \theta_2 + \theta_3 + \theta_4)$$

V_{Xi} and V_{Yi} represent the effector velocity for a unit change in a specific joint angle on the X and Y axes for a given robot configuration (θ_1, θ_2, θ_3, θ_4). In 3D, a V_{Zi} array is

introduced and represents the joint velocity along the Z axis and is found in the same manner. The fuzzy process is applied on every joint, therefore, for the 4-DOF manipulator arm, this means that there are 8 pairs of inputs ([dx] with [V_{X1}, V_{X2}, V_{X3}, V_{X4}] and [dy] with [V_{Y1}, V_{Y2}, V_{Y3}, V_{Y4}]) to study: the change in angle is calculated for each combination until the desired position is reached.

4.3.2.3. Generalization of Nedungadi's approach

In order to generalize the approach of Nedungadi [22], some adaptations are proposed, especially in the calibration of fuzzy parameters. Fuzzy logic controllers are easy to use and give efficient and accurate results, however, these fuzzy parameters need to be calibrated. Since we want the proposed approach to be general enough to deal with an arbitrary robot configuration, fuzzy parameters should remain auto-calibrated regardless of any change in the robot structure or the working environment. Those parameters are the distance between the fuzzification and defuzzification function centers.

During the fuzzification process, the width of the fuzzifying function, shown in Figure 26, is different for the displacement and the end effector velocities.

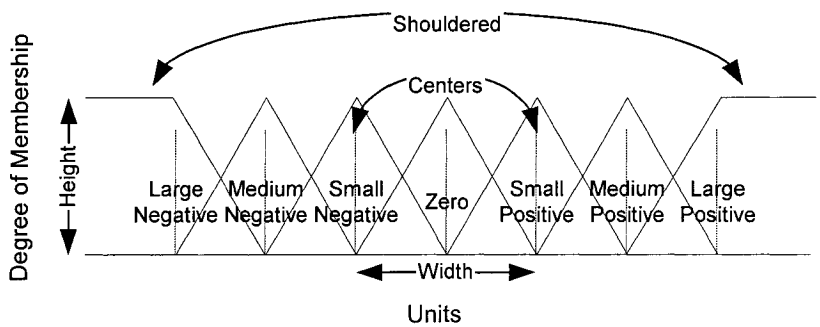


Figure 26: Fuzzy membership functions

Since we split the trajectory into smaller steps, the displacement should be optimized for movements smaller or equal to those steps. For end effector velocity, this value is usually much larger. We have defined the width of the fuzzifying function to be the distance that the effector will move in x, y and z for a joint movement of 1 radian. Determining an

optimal value of the fuzzy parameters can lead to a more efficient fuzzy controller. The most advantageous width of the fuzzy functions for joint velocity is found through experimentation to be proportional to the size of the environment, as shown in Equation 6.

$$width_{fuzzy} \approx \frac{width_{environment}}{number\ of\ fuzzy\ functions} \tag{6}$$

On the other hand, the defuzzification process was chosen to be simple. There are numerous output representations, but the simplest and quickest is to use Dirac functions also known as singletons, as shown in Figure 27. Hence, only the distance between the spikes needs to be calibrated. A large distance will yield large movements and large oscillations around the target position, while smaller distances will avoid these overshoots but may reach the position in a higher number of iterations. Testing has revealed, that for most system configurations, the optimal range of distances connecting the singletons is $angle_{out} \in [0.8, 1.1]$ degrees. This small number helps to limit large changes in joint values for most robot environments, thus providing a smooth trajectory.

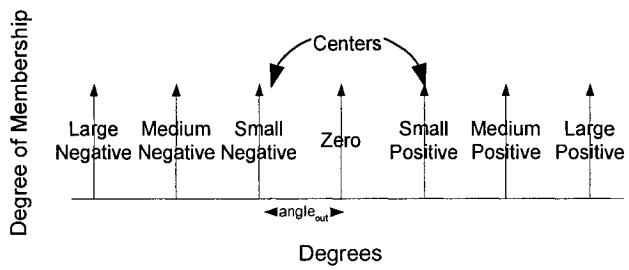


Figure 27: Defuzzification membership functions

Fuzzy logic remains a numerical approach to solve non-linear problems; therefore, it may not always converge towards a solution. The following aspects have been implemented to further refine the behaviour and increase the convergence of the system towards a solution.

- Splitting the effector path into smaller steps, defined as "intermediate positions", therefore increasing the chance of avoiding obstacles. These intermediate positions will follow the path of the end effector that have been found in the global path planning phase. As a consequence, the fuzzy logic controller is optimized within this small displacement space which gives it more stability independently of the total travel distance.

- The fuzzy controller iterates until the intermediate position has been reached. An efficient number of iterations has been found to be between 20 and 40 in order to maximize the convergence probability while eliminating useless iterations where convergence cannot be reached. 20 iterations are used as the maximum number of iterations in the test cases of Chapter 5.

- If the fuzzy logic controller exceeds the maximum allowable number of iterations, as determined previously, a deterministic approach to the same fuzzy controller is implemented to "push" the robot towards a goal position, where the fuzzy controller can start over again. This step executes one iteration of the fuzzy controller but with a much smaller variation in the joint angle output. This update to the joint angles will move the joints in the required direction but will only cause a very small movement. This represents a deterministic approach of the fuzzy controller to cause small changes.

Other parameters must be provided in order to have a better representation of the robot. An angle checking procedure is introduced to restrict the robot from folding around itself: going from an angle of 170° to 190° will most likely cause a joint to come in collision with some part of the structure. Therefore, as joint limits must be imposed, they are set so that all the joint angles can be contained within the limits provided by the user. In the test cases of Chapter 5, the joint values are contained within [-179° and 179°]. Although this is not representative of a normal robotic system, it prevents the robot from folding around itself and can be changed as these values vary from one robot to another.

As mentioned previously, Nedungadi's approach does not consider the presence of obstacles in the environment. Therefore, apart from refinements to the original method to make it more general, the present research also explores ways of combining the control on robot configuration with repulsive forces generated by obstacles in an integrated fuzzy logic controller to safely guide the robot arm in a cluttered environment. The following section describes this major improvement.

4.3.3. Collision avoidance

Safe path planning implies to move the robot up to a desired target position while avoiding collisions between the robot and the environment. The approach proposed here also considers the effects of the environment on the entire robot structure while moving the joints. To achieve this, a supplementary component is added to the fuzzy logic model. One of the inputs to the system that is considered is the probabilistic encoding of the occupancy of the environment. This data provides a relationship with the distance to a given point of a nearby obstacle as they increase for points that are closer to one. These occupancy levels can also be interpreted to be forces that are exerted away from obstacles. Using them in combination with the regular "mechanical" forces implied by the robot kinematic constraints, an approach is developed to guide the robot in the environment.

The main problem that exists in any kind of approach using repulsive forces is that it may lead the robot to be trapped between contradictory forces of the same magnitude but working in opposite direction, this is known as the classical "local minimum" problem. Through the proposed algorithm and correction methods, some of these classical problems are overcome. A system diagram of the proposed procedure including collision avoidance is shown in Figure 28.

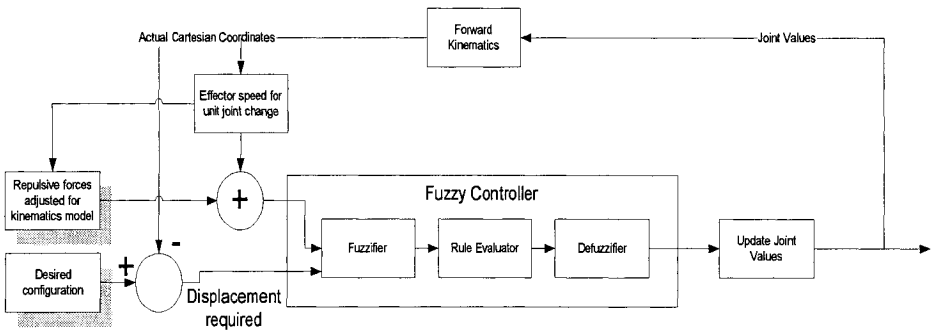


Figure 28: Overview of fuzzy controller

Repulsive forces are proportional to the probability of occupancy determined by the probabilistic model, as defined previously in section 2.3. The strongest repulsive force is found where the probability of occupancy is the highest at any point on the robot structure. In order to avoid collisions and to reduce the joint displacement when a collision is near, the repulsive forces and the joint velocities are combined. Once the repulsive force reaches a threshold, defined in the following section, the value of the velocity is slowly reduced by the force to provoke a sign change, therefore causing the joint to move in the opposite direction.

The probability of occupation is analyzed for the entire link of the robot at every step of the fuzzy process; this includes checking for collisions during every iteration of the fuzzy logic controller. While this increases computation times, is it a crucial step to avoid collisions during robot movement, and determining the regions in the robotic structure where the probability of the presence of an obstacle is the largest (therefore is at the most risk of colliding with objects) is essential.

Figure 29 displays the block diagram of the proposed algorithm.

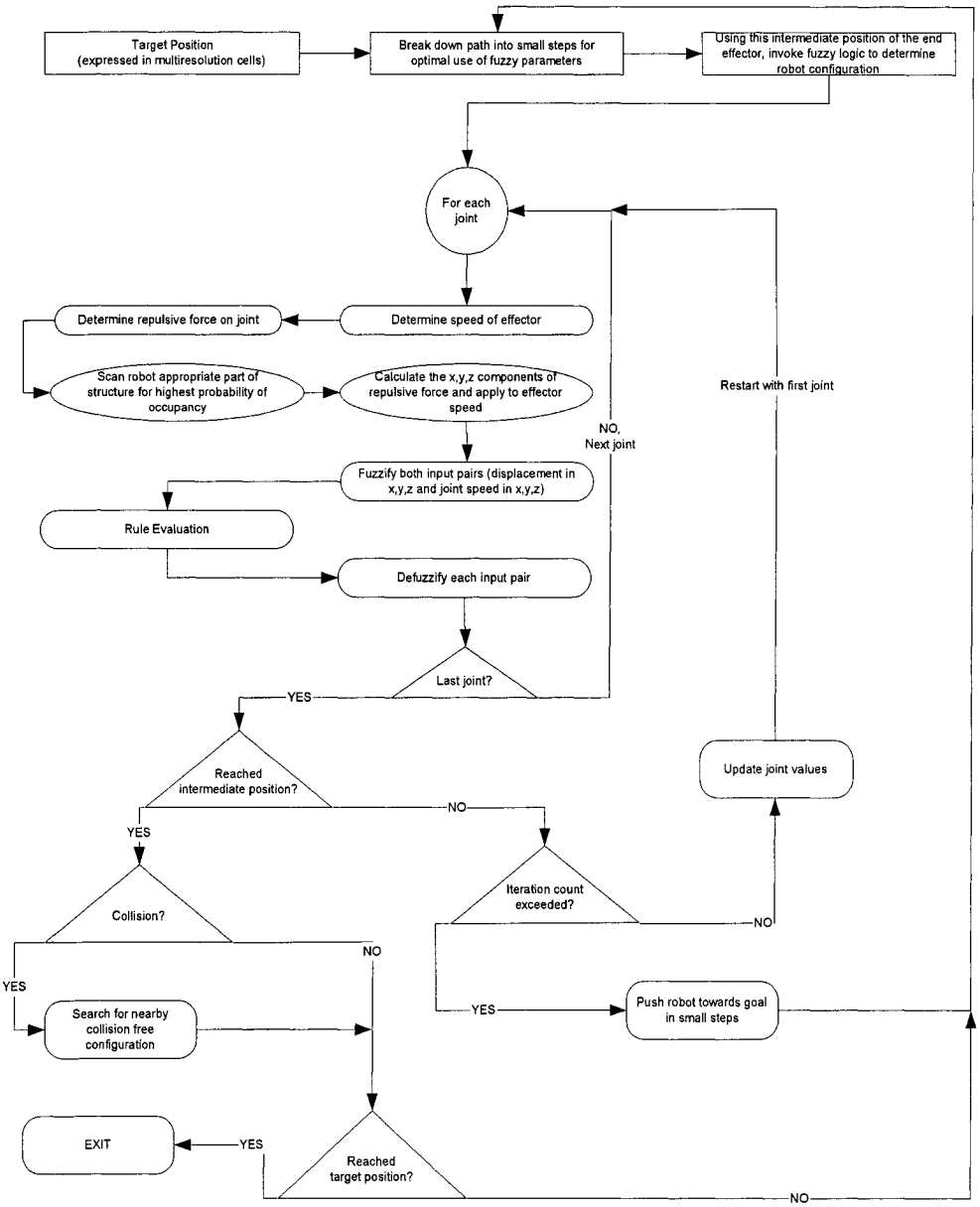


Figure 29: Block diagram of fuzzy logic/collision avoidance procedure

4.3.3.1. Repulsive field and forces

As stated in section 4.3.2.2, the fuzzy controller has two types of inputs: the displacement distance and the effector speed with respect to every joint movement. In order to include the repulsive forces, those speeds are modified if the robot is close to an obstacle: we wish

to reduce (or change) the movement of the robot when a joint is near an obstacle. As a joint comes closer to an obstacle, the repulsive force must slowly force the joint to move in the other direction. The rest of the structure will, in turn, overcome this change by adjusting in the following iteration or by exploiting the redundancy of the manipulator, if available. The repulsive force is proportional to:

- The probability of occupancy provided by the occupancy map of space $p[\alpha c]$, which is converted to a functional value $F_{unjusted\_rep}$ using Equation 7,
- The projected repulsive force on the x, y and z axis to consider the angle of the force, as shown in Figure 30, and
- A force coefficient, αeff_{rep_force}.

This coefficient is calibrated to represent the occupancy value at which the robot is deemed too close to an obstacle, and the corresponding joint needs to move in the opposite direction. The coefficient needs to be large enough to make sure that the robot will be pushed away sufficiently if close to an obstacle, but small enough not to cause large swings in the opposite direction or to reduce the working space of the robot. The coefficient is calibrated from the maximum acceptable repulsive force and is auto-adapted based on the size of the environment. It represents a threshold at which a collision is imminent and the joint should move in the opposite direction.

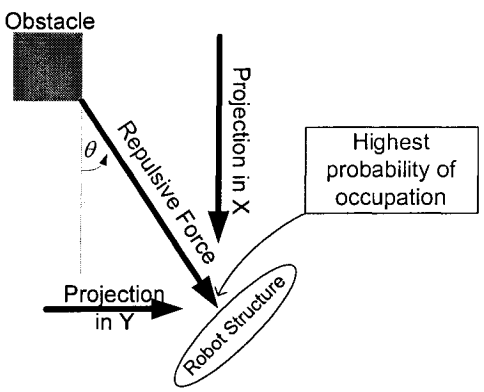


Figure 30: Projection of repulsive forces for a planar robot

The force exerted by the obstacles is proportional to the occupancy probability found in a given cell where the robot lies. In order to use it to push the robot away, we need to convert it to a value that can be useful and stay independent of the size of the probabilistic data, it is based on these criteria:

- The repulsive force must be zero when the probability of occupation is null.
- The repulsive force should converge towards 1 as the probability increases.
- A threshold should be established which will act as the limit to which the robot can approach an obstacle.

For the sake of simplicity and to lower memory requirements, the probability of occupancy, for our experiments, is encoded as an integer between 0 (no obstacles) and 16 (probability of 100%). Equation 7 shows the mapping between the discrete probability occupancy and an unadjusted repulsive force in the proposed method. This expression results in negligible repulsive forces when the probability is small while it imposes a severe repulsive force when approaching obstacles.

$$F_{unadjusted\_rep} = \frac{p[occ]}{3 + p[occ]} \tag{7}$$

Figure 31 illustrates the output force as a function of the probability of occupation.

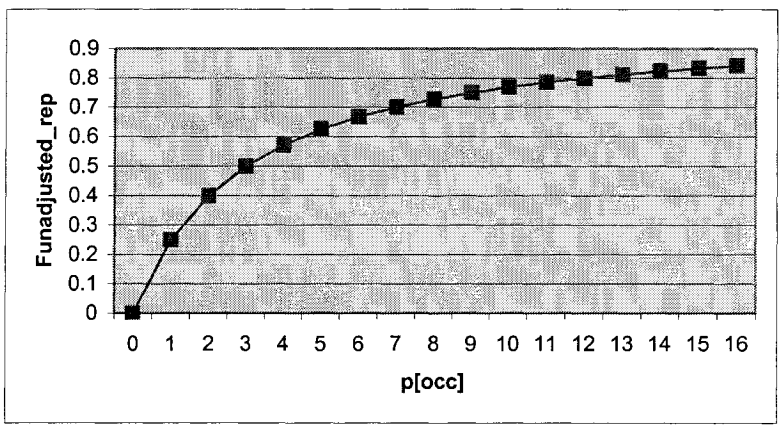


Figure 31: Repulsive forces as a function of probability of occupation

The effects of the repulsive forces on the fuzzy logic controller are to force the joint angle to move in the opposite direction of the obstacle when a collision is deemed imminent. This threshold is determined by the force coefficient. At a probability of $F_{unadjusted\_rep}=coeff_{rep\_force}^{-1}$, we wish the joint movement in the direction of the obstacle to be stopped, when this threshold is exceeded, the updated joint angle will cause the robot to move in the opposite direction. A value of $coeff_{rep\_force}=2.0$ is chosen since it has been found experimentally to provide a good compromise between the possibility of collisions and manipulator dexterity.

The overall repulsive force in 2D is therefore computed following Equation 8.

$$\begin{aligned} \vec{F} &= \vec{F}_{unadjusted\_rep} * coeff_{rep\_force} \\ F_x &= \vec{F} * \sin(\theta) \\ F_y &= \vec{F} * \cos(\theta) \end{aligned} \tag{8}$$

In order to ensure that the fuzzy controller will never cause the robot to be in collision with an obstacle, a supplementary correction method has been introduced. If during the defuzzification process a configuration is returned that causes the robot to be in collision with an obstacle, this method is invoked and searches for nearby robot configurations that avoid collisions with obstacles until the robot is in a safe configuration where the fuzzy logic controller can continue. Although inefficient but seldom used, this ensures that the robot structure will never be in collision with an obstacle.

4.3.3.2. Other improvements to the original approach

This section defines and elaborates on issues regarding collision avoidance and the generation of repulsive forces. It contains algorithms and fine-tuning approaches developed to improve path planning and tracking and achieve quicker convergence of the fuzzy logic controller to a valid collision-free configuration.

Every link must feel the effects of the repulsive forces exerted on the robot structure by nearby obstacles and must adjust accordingly. The first assumption that we make is that the

starting position of the robot is not in collision. At some point during the trajectory of the robot, some part will encounter an object. The logic behind the implementation is to move the joint at which there is a collision away from where the object is. This may lead the robot to be in a different position than the one desired, however, the purpose of this is to place the arm in a better situation (or position) to approach the target position, while avoiding a collision. The following iterations of the fuzzy logic will consider these corrections and eventually reach the desired position. This section details various strategies to apply the effects of the repulsive force that have been investigated in order to prevent collisions.

In the first case, the algorithm was configured to simply move the problematic joint in the opposite direction: this approach is not very well suited since if there is a collision near the joint, it may have to be moved by a large amount. The scenario is displayed in Figure 32a.

Second, once a collision is imminent at a link, the robot was to move the previous joint in the opposite direction, as shown in Figure 32b: this approach worked slightly better than the previous one, but the resulting angle displacement tends to be too large and causes a reduction in the dexterity of the manipulator. Experimentation with different scenarios has been performed such as a joint being responsible for a region near the joint itself. Good results have been found when checking the regions both prior and after the current joint.

In another implementation, the robot was to "curl" around the obstacle, as shown in Figure 32c: to move the preceding joints in the opposite direction and the joint in question in the original direction. This approach demanded too much calibration (such as the displacement of the subsequent joints), was too dependent on the robot architecture and was inclined to bring the robot to be too close to the obstacles.

Finally, the controller caused the robot to move all the previous joints in the structure to avoid the obstacle, as shown in Figure 32d. However, as the joint is further to the one in collision, the correction can be smaller: this approach yielded the best results, and caused the fuzzy logic controller to converge towards a valid solution more rapidly. In this strategy, the

smaller movements caused by the previous joints help bring the end effector towards a safer position to increase its dexterity. This approach also has been tested with different variations with mixed results, such as trying to "fold" the robot so that it compresses towards the safe zone.

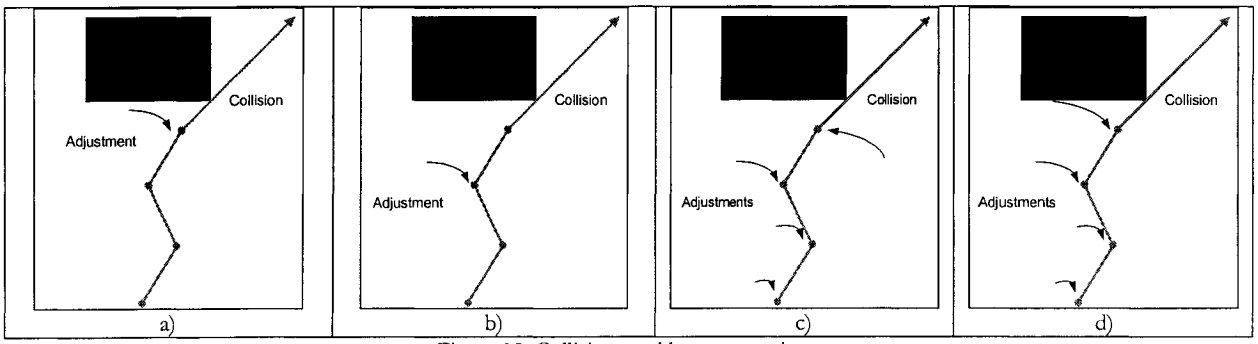


Figure 32: Collision avoidance scenarios

The latter option has been implemented: all the joints preceding the joint near a collision feel the repulsive force. For any given joint, the following procedure is followed:

- Scan the robot structure to determine the position in the robot that has a higher probability of occupancy in all the joints following the one under examination. For instance, if we analyze joint 3, we want to move joint 3 as well as joints 1-2.
- The repulsive force is computed and is a function of the probability of occupation, the distance of the joint to the collision and the force coefficient, $coeff_{rep\_force}$.
- This value constitutes an input to the fuzzy logic controller and represents the repulsive force that the joint will experience during this iteration of the fuzzy logic control.

The updated fuzzy input for joint number i with end effector velocities of V_x and V_y and where the highest probability of occupancy $p[occ]$ is located in joint j is shown in Equation 9 which then becomes the inputs to the fuzzy logic controller. Since only the following and no preceding joints are examined, $i \leq j$.

$$V_{Xi\_updated} = V_{Xi}\left(1 - F_x * \frac{i}{j}\right) = V_{Xi}\left(1 - \frac{p[occ]}{3 + p[occ]} * coeff_{rep\_force} * \sin(\theta) * \frac{i}{j}\right)$$
$$V_{Yi\_updated} = V_{Yi}\left(1 - F_y * \frac{i}{j}\right) = V_{Yi}\left(1 - \frac{p[occ]}{3 + p[occ]} * coeff_{rep\_force} * \cos(\theta) * \frac{i}{j}\right) \quad (9)$$

The approach presented in this section can easily be applied to bulkier robots. The repulsive force on a joint is determined where it is greatest in the structure, regardless of robot architecture. The repulsive force exerted is computed by determining the largest probability of occupancy of any position in the joint's zone of influence.

4.4. Comparison of methods

The fuzzy rule base proposed by Nedungadi [22], shown in Table 4, provides a good basis for robot manipulator control but, it also possesses some drawbacks. Most notably, after the effector speed is fuzzified, only a change in the sign of the effector speed (input 2) provides a change to the output value. When using the repulsive force to adjust this speed to provide collision avoidance, we need to improve the proposed rule base such that a change in the magnitude in input 2 can provide a different output value. Also, the rules proposed by Nedungadi are not representative of the input values: the largest output should be when the required displacement value is large and the effector velocity is small (Input1 is NL or PL and Input2 is NS or PS). A refined version of the fuzzy rule base is then proposed as shown in Table 5.

		Input 1=effector displacement required						
Input 2=velocity of effector for positive joint change		**NL**	**NM**	**NS**	**Z**	**PS**	**PM**	**PL**
	NL	PM	PS	PS	Z	NS	NS	NM
	NM	PL	PM	PS	Z	NS	NM	NL
	NS	PL	PM	PM	Z	NM	NM	NL
	Z	Z	Z	Z	Z	Z	Z	Z
	PS	NL	NM	NM	Z	PM	PM	PL
	PM	NL	NM	NS	Z	PS	PM	PL
	PL	NM	NS	NS	Z	PS	PS	PM

Table 5: Refined fuzzy association banks

A major change that is implemented in this new rule base is that we want to limit the joint movement when the speed is high, for example, in the corners of the table, if the

displacement and movement are large, the resulting angle change should not be as large as proposed by Nedungadi. Instead, a medium displacement is proposed. Similar reasoning can be extended to other cases like [NL PL], [PL NL], [PL PL], [NL NM], [NL PM], [PL NM], [NP PM], [NS PS], [NL PS], [PS NS] and [PS PS].

In order to evaluate the performance of the refined fuzzy association bank, a comparison was done for path planning and path tracking in an empty environment. The updated fuzzy association bank is designed to cause better convergence in the fuzzy logic controller in the presence of obstacles. It has also been improved to include a better representation of the kinematics of the robot. Path tracking for both fuzzy association banks was tested in an empty environment for various target positions. A comparison of computation times is shown in Table 6.

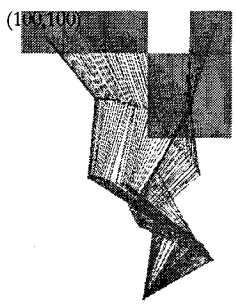
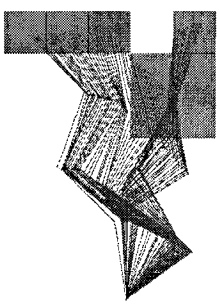
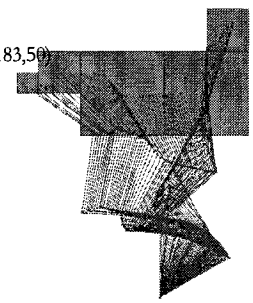
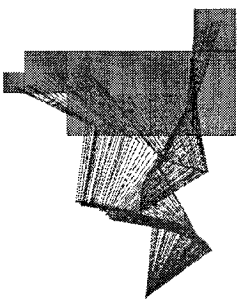
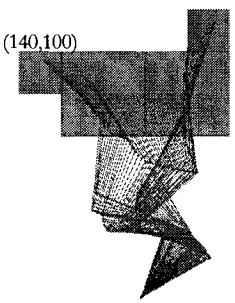
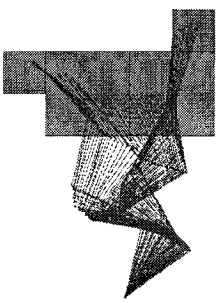
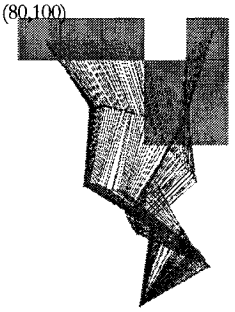
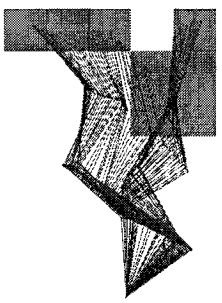
Target position	Computation times (seconds)	
	Original banks	Refined banks
(100, 100)	17.550	10.790
(183, 50)	4.086	5.948
(140, 100)	3.565	4.877
(80, 100)	36.342	14.020
(100, 410)	1.702	1.762
(100, 250)	2.093	2.323
(188, 480)	6.800	6.660

Table 6: Comparison of fuzzy association banks

Comparable computation times were found with the refined version in empty space, as proposed in this work. Generally, it was noted that the original banks provided slightly faster computation times when the robot is moving near the middle of the workspace while the refined algorithm provided better results elsewhere. This is particularly true in situations where the arm is near full extension; the original banks were roughly twice as slow to converge towards the target position.

Additionally, when used in cluttered environments, such as those in Chapter 5, the refined table provided much better flexibility in the presence of obstacles. As Table 7 shows, the trajectories using both banks are very similar. While the new path tracker is designed for

experimentation in cluttered environments, it provides performance in an empty workspace that is comparable to that of Nedungadi.

Target position	Original banks	Refined banks
(100, 100)		
(183, 50)		
(140, 100)		
(80, 100)		

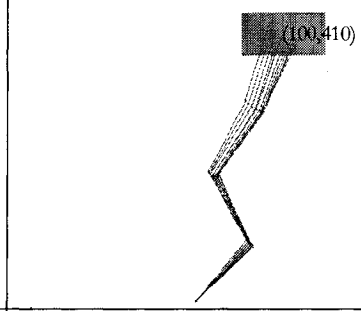
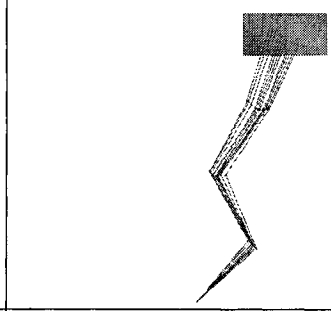
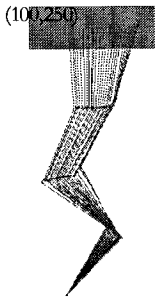
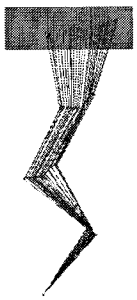
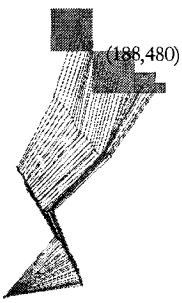
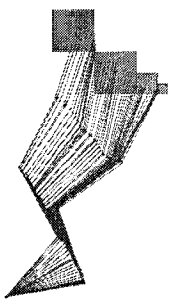
(100, 410)		
(100, 250)		
(188, 480)		

Table 7: Comparison of path for fuzzy association banks

4.5. Conclusion

In this chapter, the updated fuzzy logic controller is developed to include the repulsive forces of nearby obstacles. The controller is designed for use without changing the internal parameters therefore making an easy transition between different test applications. In Chapter 5, a set of experimental testbeds are developed and analyzed using the proposed strategy. The results confirm the robustness of the approach for both 2D and 3D workspaces.

CHAPTER 5: EXPERIMENTAL RESULTS

5.1. Introduction

In this chapter, the experimental testbed used to evaluate the performance of the proposed collision-free path planning approach is detailed based on the methods detailed in Chapter 3 and Chapter 4. Results from various scenarios and manipulator architectures are presented and analyzed for 2D and 3D workspaces.

5.2. Experimental setup

The simulations shown in this section were executed on a 450 MHz SGI machine with 218 MB of RAM. The results come from experimental observations collected from a software implementation of the concepts defined previously. 2D and 3D visualization tools have also been designed to provide visual display of the resulting paths and facilitate tuning and analysis.

The main program reads and interprets the input data encoded as either images or binary files, computes the various potential fields, encodes the robot working environment and determines the manipulator path. The program has been implemented using standard C libraries with Microsoft Visual C++ 6.0. In order to provide a thorough understanding of the steps of the program, most of the functions were custom-built. Functions of similar concepts were regrouped together in distinct libraries: input/output data manipulation, probabilistic operations, quadtree or octree manipulation, global path planning and local path planning methods. This provides greater flexibility when changes are required.

The main application also displays run-time information for every step throughout the execution: reading/writing data, n-tree operations and path planning steps. During path planning, the position of the robot and the corresponding joint values are displayed at every step to provide the user with a detailed knowledge of the situation. Execution times can be determined and displayed for every step to provide comparative results and interrupts have been implemented for the user to stop, pause or create intermediate data files to visualize

the behavior of the algorithm and display information during the execution of the program. Almost all of the concepts presented in this research can be saved for later visualization, including the probabilistic data, the global path, the repulsive field, and the attractive field. This console application is nearly identical in 2D and 3D implementations. In the latter case, the console data differs only by displaying the third Cartesian coordinate. A sample screen shot is of this application is shown in Figure 33.

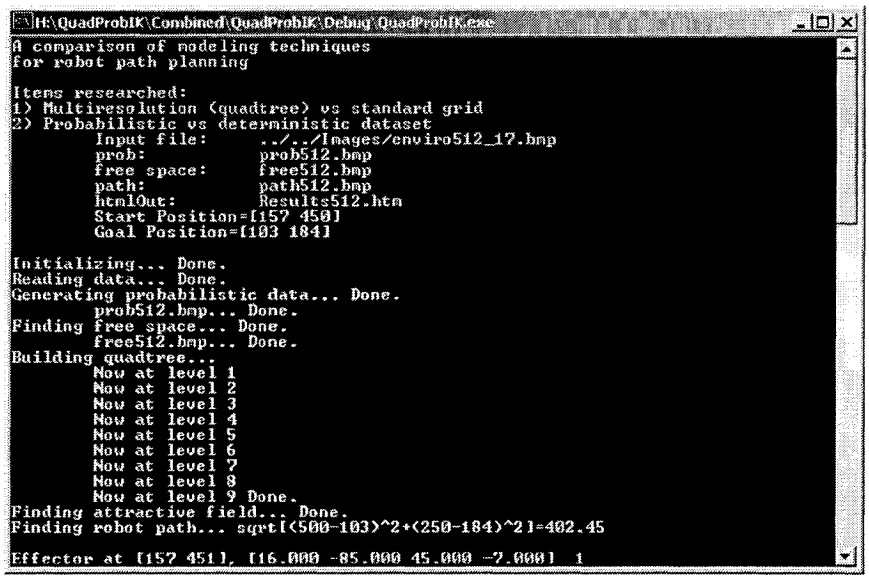


Figure 33: Sample console implementation

In order to meet one of the goals of this research, the algorithm was implemented for use with many different manipulators. Therefore, only the forward kinematics parameters and its first partial derivatives, to represent the end effector velocity, are encoded. The forward kinematics and end effector velocity equations are found through the symbolic MATLAB toolbox and added to the main application as a series of equations providing the necessary relationships stored in an array.

For 2D environments and planar manipulators, the workspace data can be stored in two different ways. The data can be stored in text files containing environment information and manipulator configurations. However, the results displayed in this manner are not easy to analyze. The second method uses image files that can display both environment information

and manipulator configurations. A codec was built that follows the typical standard format for bitmap images, as detailed in Appendix C. These image files can also be loaded onto another custom or commercial application for visualization.

Two different visualization schemes are used for 2D environments: a first implementation generates a web page that contains relevant information and loads the images generated. The second implementation was developed in visual C++ to provide more flexibility for parameter changes, such as start and goal positions and joint values. The results shown in this section come from the latter option. This option is detailed in Appendix D.

Also, a console application was implemented to provide analysis for the forward and inverse kinematics of the robot. It was used to calibrate the fuzzy logic controller, determine initial joint values and test the limits of the robot arm.

For 3D workspaces, creating and modifying environments is more complicated, therefore an application was implemented to manipulate environments. The environment, with obstacles and robot, can be displayed on screen as well as rotated and translated along all three axes; this also provides the capability to zoom in and out of particular situations. The application can also be used to validate the forward kinematics of the robot by loading angles to visualize the behavior of the robot. The 3D results shown in this section come from this Visual C++ application that is also detailed in Appendix D.

5.3. Environment configuration

The workspaces used for experimentation consist of an environment usually containing a "safe" zone near the robot base and a number of obstacles elsewhere. The obstacles are either placed randomly or in a position to research the effects of particular situations. Various tests were conducted to explore the behavior of the algorithm when facing situations such as obstacles in close proximity of robot, narrow corridors and moving the manipulator around an obstacle.

In 2D, the environment is represented with a greyscale image that can be drawn from any drawing program, such as MSPaint, and saved with the ".bmp" extension (refer to Appendix C). The typical size of the image used is 512x512 although different formats have also been tested. This environment file can contain either probabilistic data or deterministic information which can be transformed into probabilistic representations. A sample environment is shown in Figure 34.

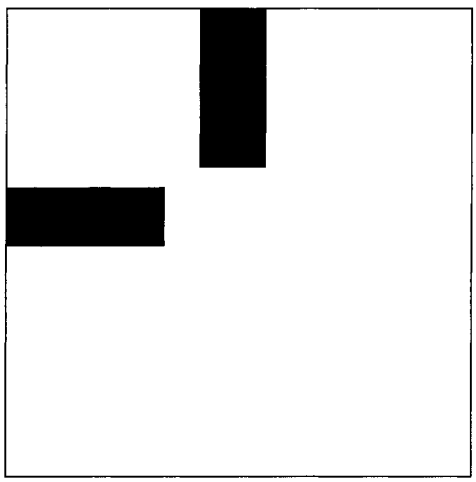


Figure 34: Sample 2D environment

In 3D, a simple program developed using C++ and OpenGL allows the user to load and display the environment, as well draw new obstacles and save to a tab-separated text file. The typical environment sizes are 64x64x64 to keep things tractable since those models are created manually.

5.4. Experimentation

The proposed path planning method has been tested with robots having with a various numbers of degrees of freedom and with different architectures to evaluate its flexibility, performance and robustness. In this section, the simulation results shown come from two different robot types: generic 3 and 4 degrees-of-freedom planar robots and a 3D PUMA-like robot both containing only revolute joints.

The Denavit-Hartenberg parameters for the planar robots are shown in Table 8.

Joint	L_i	D_i	α_i	θ_i
1	L_1	0	0	θ_1
2	L_2	0	0	θ_2
3	L_3	0	0	θ_3
4	L_4	0	0	θ_4

Table 8: DH parameters for 2D planar robot

The link lengths, L_i, are left as variables to provide greater flexibility when changing robot architecture and are encoded as a ratio to the size of the environment. For the 3 degree-of-freedom planar robot used, we set $L_4=\theta_4=0$. In Table 9, the Denavit-Hartenberg parameters for the 3D PUMA-like robot are displayed, as inspired by[41].

Joint	L_i	D_i	α_i	θ_i
1	0	D_1	Pi/2	θ_1
2	L_2	0	Pi/2	θ_2
3	0	D_1	Pi/2	θ_3
4	L_4	0	Pi/2	θ_4
5	0	D_1	Pi/2	θ_5
6	L_6	0	0	θ_6

Table 9: DH parameters for 3D PUMA-like robot

5.4.1. 2D test cases

In this section, the results of a number of different 2D test scenarios are displayed and analyzed. The robot in use for these simulations consists of 3 or 4 degrees of freedom, and of various joint lengths.

5.4.1.1. First test case

In the first test case, the base of the robot is fixed in the middle of the workspace near the bottom; [500 250] for an image of size 512x512. The start position is chosen arbitrarily at [83 363] and the target position is also chosen arbitrarily as long as it is contained in the enclave seen in the top left corner, in our case, we have used [100 100]. Figure 35 displays this environment with the robot in its starting position. The obstacles are depicted in black while the free space is in white.

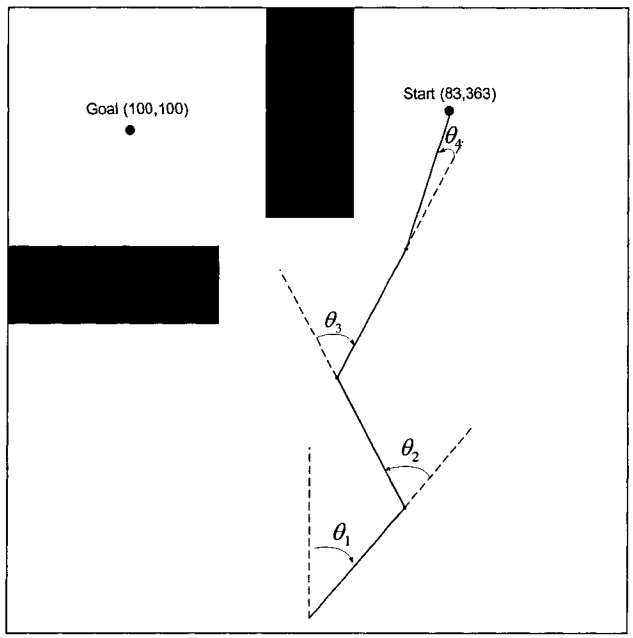


Figure 35: Initial robot configuration

From the starting position, the free space is encoded into a sequence of multi-resolution cells. The attractive field is then computed on the quadtree sequence following the neighbor search techniques introduced in Chapter 3 and the global path planning method, detailed in section 4.2, is applied. To guide the path planner, a sequence of neighbor cells are selected by taking for each step the neighbor cell with the smallest distance to the target position. The resulting path is a sequence of multi-resolution cells linking the start and target positions, within the security margin but which may overlap the probabilistic data. The path of the effector is then interpolated to reach the middle of the multi-resolution cells which are depicted in grey in Figure 36. Supplementary intermediate positions are finally computed to link the cells, either through common corners or smallest distance to the neighboring cell. Intermediate points are displayed with smaller squares in Figure 36. This path yields an approximate sequence for the end effector to follow and represents the global path of the end effector.

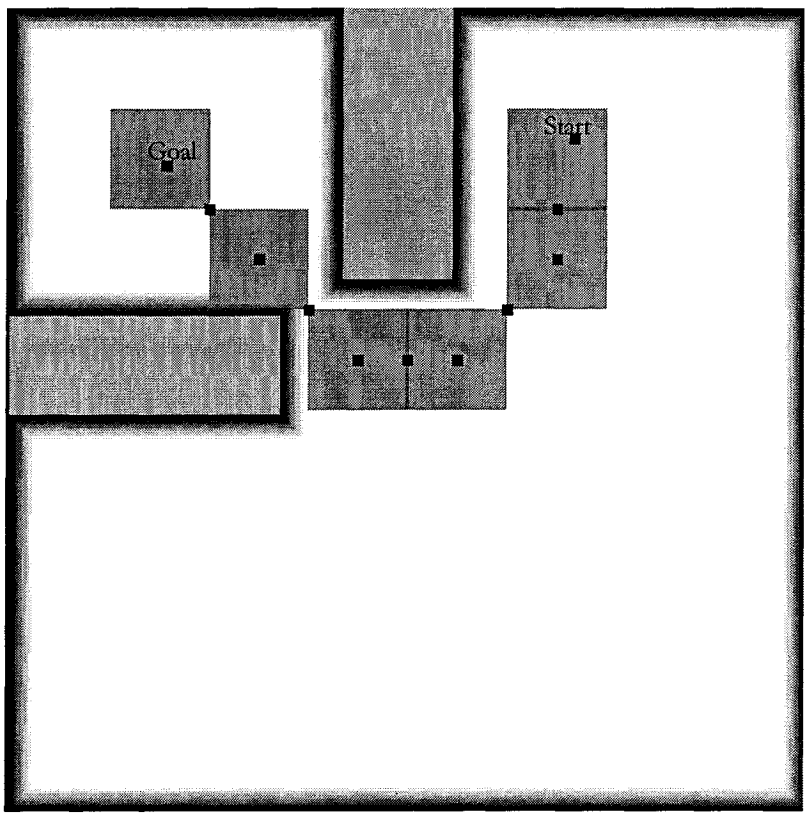


Figure 36: End effector path

Next, the global path of the end effector is refined using the fuzzy logic-based path tracking approach presented in Chapter 4 to obtain the overall path of the manipulator arm as shown in Figure 37. From the starting robot configuration, the local path tracking method is executed to determine the robot configurations from the global path that has been divided into smaller intermediate segments for which the fuzzy controller is optimized; in this example 5 pixels or about 1% of the workspace size is used. A series of joint angles is computed to reach those segments and displayed in Figure 37. The global path is also shown for analysis.

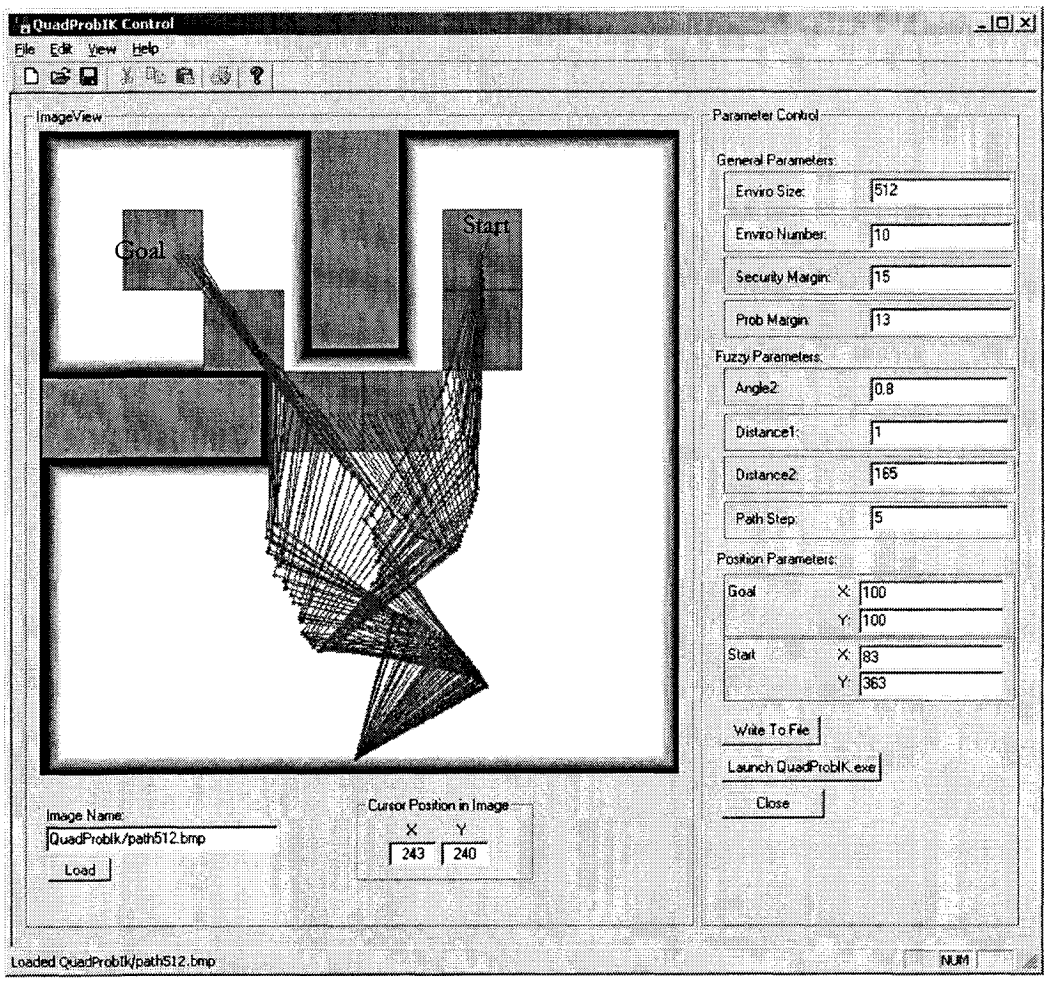


Figure 37: Complete robot path for the first test case

For the preceding figure, the joint values at every step are shown in Figure 38. One of the initial objectives was to propose an algorithm that minimizes the joints' movement between consecutive positions to maximize stability and minimize the risk of collisions that often results from abrupt displacements. By consequence, the occurrence of singularities can be greatly reduced. This is achieved, as the largest joint displacement performed between consecutive positions is approximately 4.5 degrees in this example.

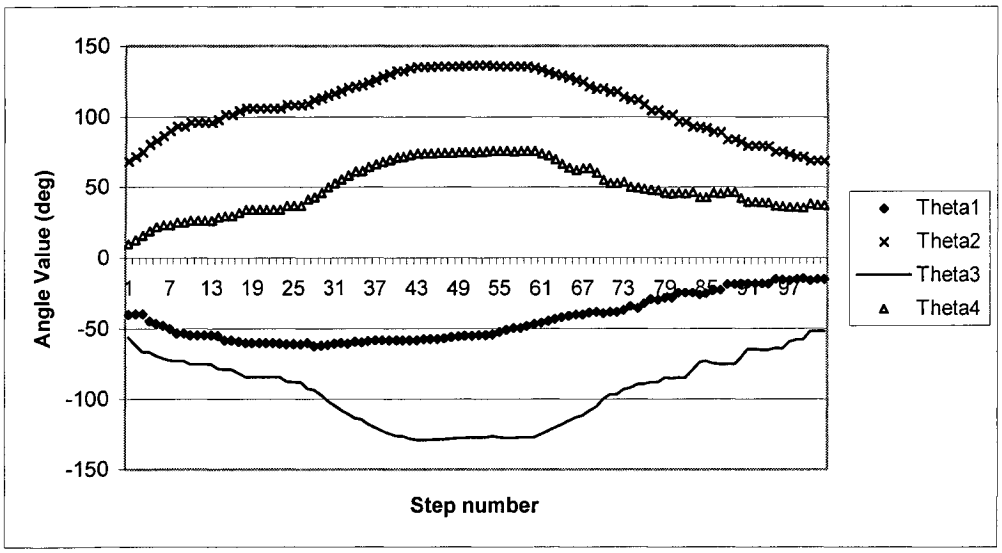


Figure 38: Angles progression for the first test case

Figure 39 displays the position of each of the joints for every step and shows that the controller brings every joint in a smooth progression along the trajectory while keeping the joint and the rest of the robot structure collision-free.

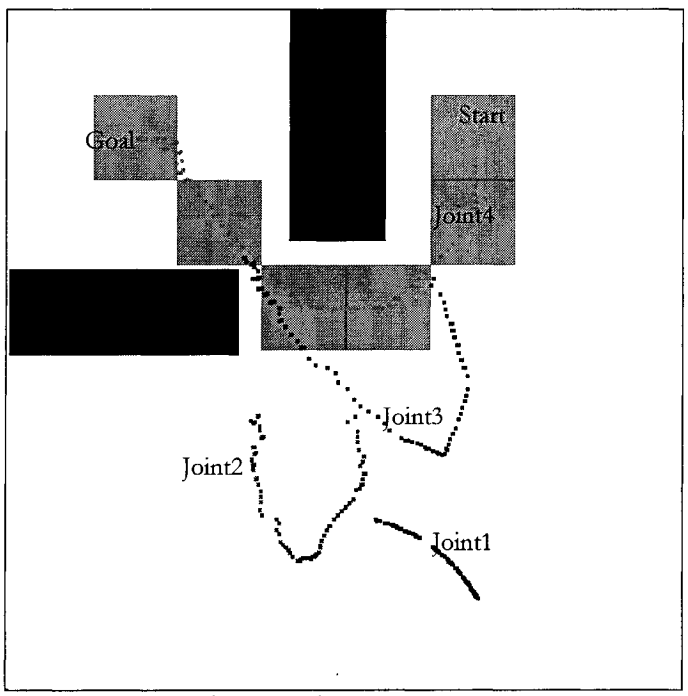


Figure 39: Joint movement

The repulsive forces felt by every joint at every step are displayed in Figure 40. The data shows only the repulsive forces when they become noticeable which represents the second half of the trajectory, when the end effector enters the opening. The average repulsive force is higher for joint 4 as it is the one most affected by the nearby obstacles. The other joints also feel this repulsive force, which attempts to move the robot towards a safe configuration that eventually leads to the target position.

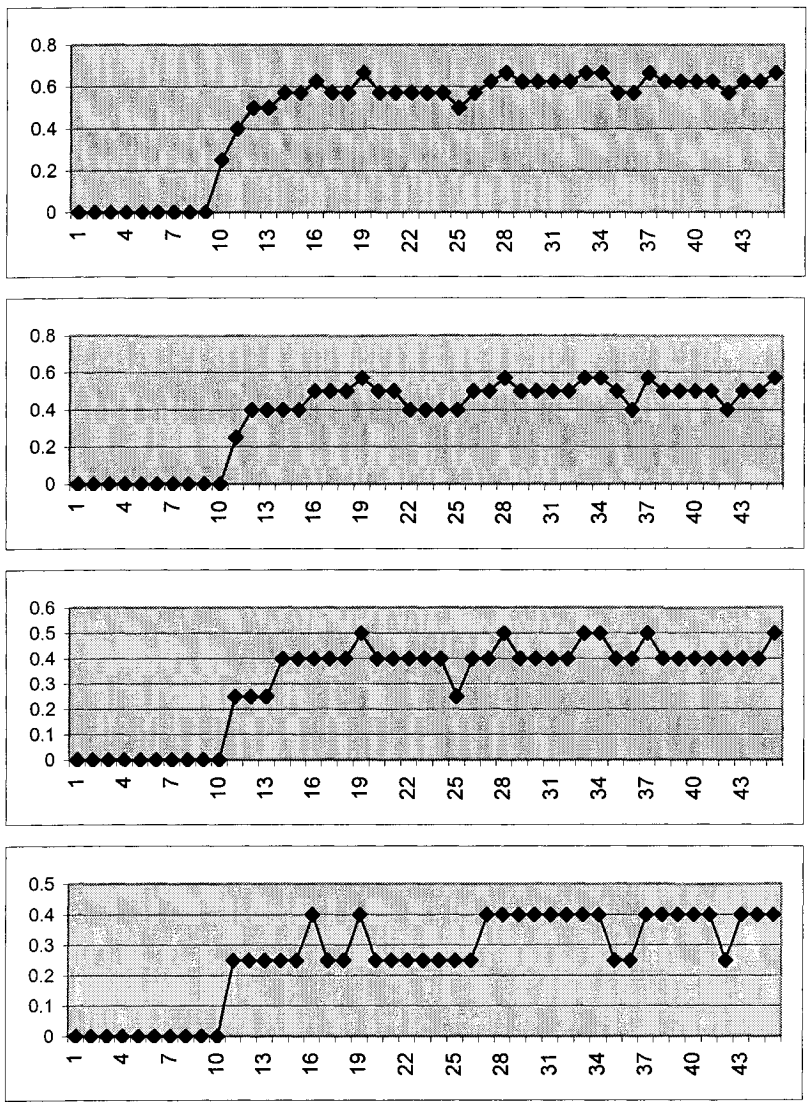


Figure 40: Repulsive forces for first test case

The current environment has been selected because it puts the robot into many interesting situations. The first part of the trajectory is used to analyze the behavior of the algorithm without strong repulsive forces. The opening between the objects shows the robustness of the algorithm in dealing with narrow corridors.

For this case, the execution time of the application is 17.040 seconds: this includes all the steps: environment input, quadtree encoding, repulsive fields computation, global and local path planning.

In this test case, there are no collisions with obstacles; the robot comes close to an object near the end of the trajectory as joints 3 and 4 are the most affected. The algorithm considers those repulsive forces and determines an adequate configuration to fold the manipulator arm around the obstacle. This reaction reveals to be successful as the resulting path is exempt of collisions.

The main factor in increasing computation time in this example is the higher number of iterations of the fuzzy logic controller to converge towards a solution. In the first half of the path, the controller determines a solution in a few iterations and computation times are low. Once the manipulator enters the opening, the fuzzy controller converges towards a solution in more iterations and the application is slowed down. At this part of the trajectory, the repulsive forces are the strongest especially on joints 3 and 4. However, a collision-free path is still found.

The algorithm always yielded good and similar results when experimenting with different starting positions in the same environment. In this test case, even with a start position near the obstacle at the top of the environment, the global trajectory brought the robot to the same multi-resolution cells on the right and the rest of the trajectory is similar. The test case has also been experimented with a smaller opening and goal position closer to the obstacles. In both cases, the fuzzy controller is able to compute a complete trajectory; however, the resulting path brings the robot closer to the obstacles.

The repulsive force scenarios explained in section 4.3.3.2 were also experimented with using this test case. Some of these methods were able to find a collision-free path but the convergence speed of the fuzzy controller was lower. The others were much more prone to be caught in a local minimum. The proposed approach that consists of moving all the previous joints yielded the overall best results.

5.4.1.2. Second test case

In this situation the robot is meant to go around a large obstacle located in the upper left corner of the environment while the robot base remains at (500,250). This example purposely illustrates the actions of the global path planning phase, as it allows to recognize two large unoccupied zones in the center of the workspace and will favor them rather then smaller regions closer to the obstacle to increase efficiency, reduce computation times and minimize the risk of collision. In this test case, the fuzzy parameters have not been changed from the first settings in order to show the robustness of the algorithm.

In this test case, the global path is quickly computed as is consists of only 4 multi-resolution cells. This helps determine a local path further from obstacle as the target path stays away from obstacles as large multi-resolution cells are generally further from them. In contrast, if a fixed-cell encoding would have been used, the global path would have been close to a diagonal line between the start and target position and would have come very close to the lower right hand corner of the obstacle near the middle of the trajectory. With the proposed approach based on multi-resolution probabilistic occupancy maps, the multi-resolution cells are rather linked in a manner to minimize the occurrence of a path near an obstacle by using the middle of the cells. These intermediate points for the end effector help in minimizing the risk of collision.

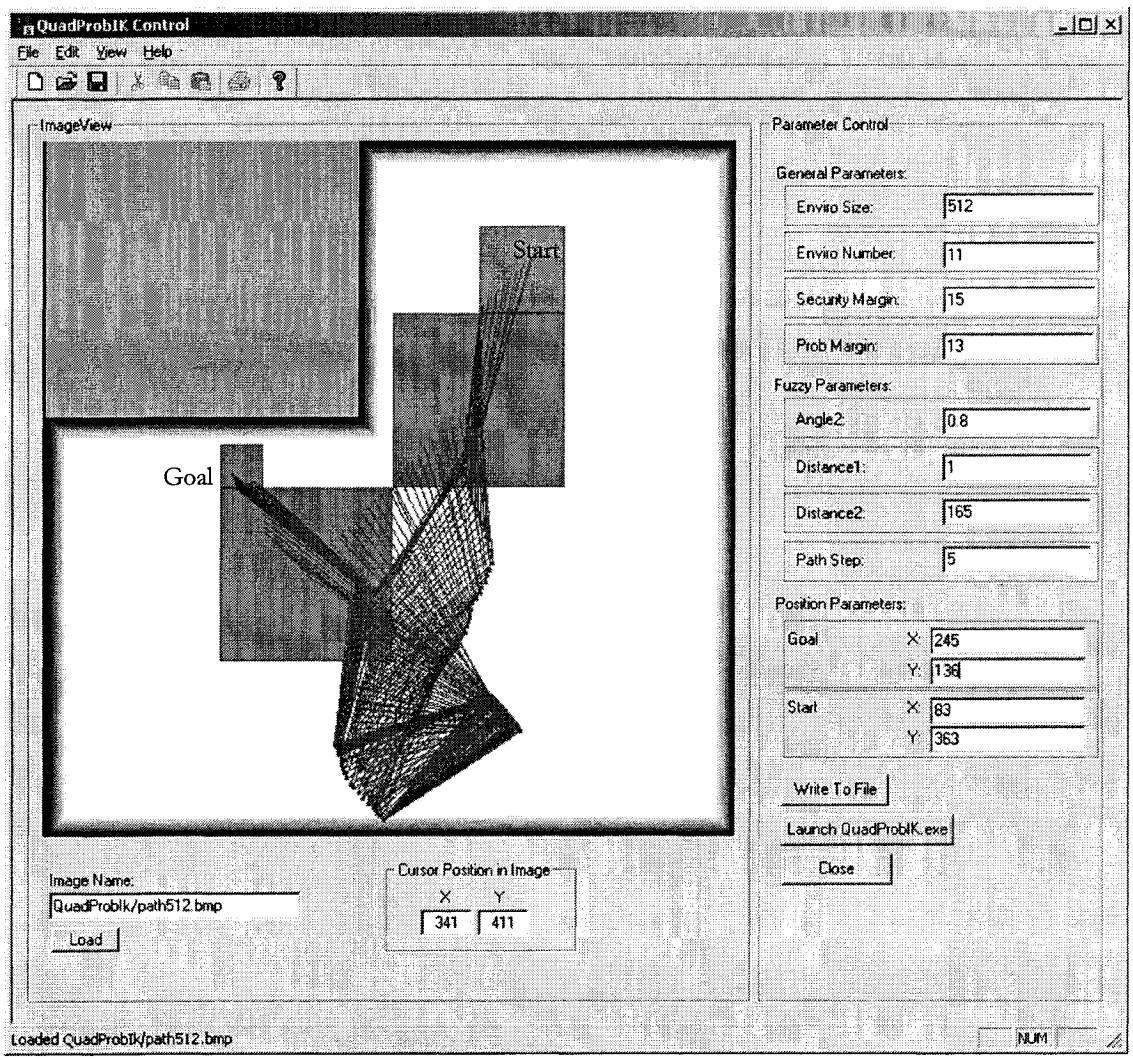


Figure 41: Path sequence for the second test case

Even though the neighbor search is significantly reduced by the limited number of cells included in the end effector's path, the computation time for the second test case is ~20 sec which is slightly longer than that of the first test case. The primary reason is the lower rate of convergence of the fuzzy controller when the robot is folded around itself. This happens when the middle of the second large multi-resolution cell is traversed. The fuzzy controller takes more iterations to converge and slows down the computation time of the program. This drawback, however, does not prevent the controller from finding a collision-free solution. The rest of the path in the presence of obstacles confirms the faster computation

when the modified fuzzy association banks are used compared with those proposed by Nedungadi [22]. Since the banks were modified in order to reduce the number of iterations in a cluttered environment, the changes are therefore important in reducing the computation times when the robot is without the influence of repulsive forces as well as under their presence. The proposed approach also computes a smooth path as the largest angle change between consecutive iterations is 3.5 degrees, as confirmed in Figure 42.

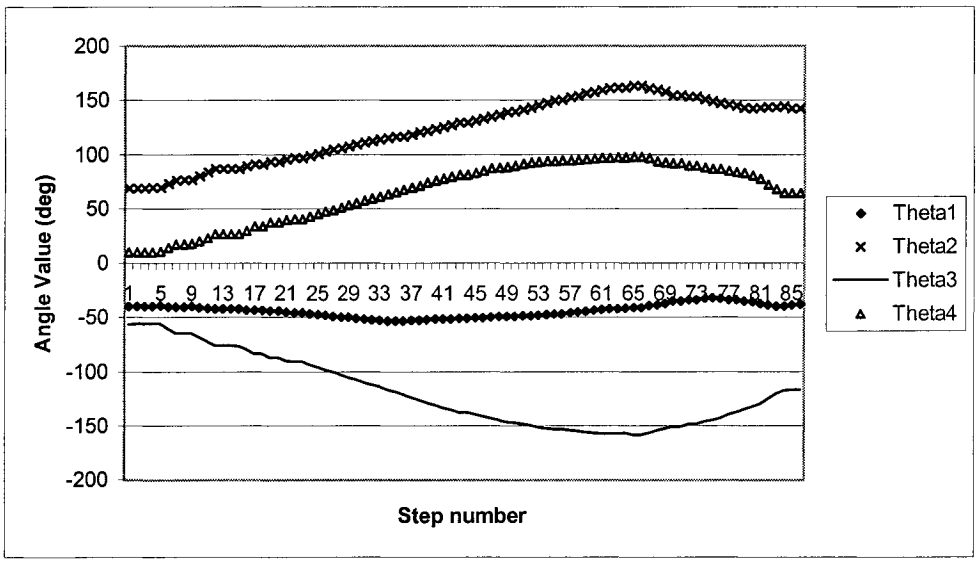


Figure 42: Angles progression for the second test case

5.4.1.3. Third test case

As shown in Figure 43, a different robot configuration is used for the third test case; every joint is longer than before with the exception of the last which is shorter.

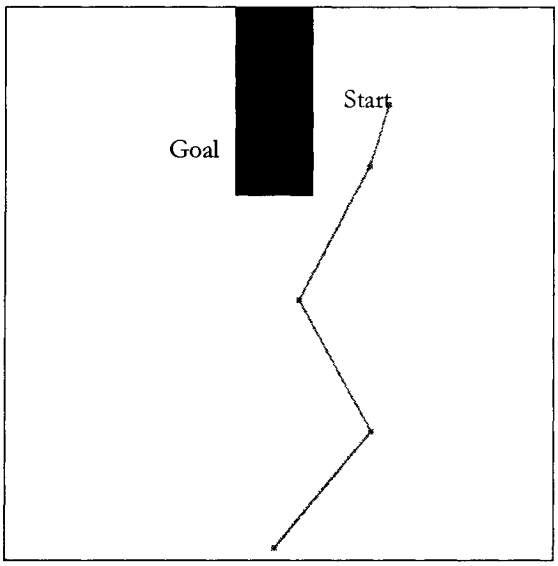


Figure 43: Robot configuration for the third test case

For the third test case, the overall path is displayed in Figure 44.

As in the previous test cases, the fuzzy parameters have not been changed in order to show the robustness of the algorithm. The results are similar to the robot used in the previous test cases. The robot navigates easily in the early portion of the path, but slows down when it is folded at the center of the large multi-resolution cell. The robot avoids the obstacles well near the end of the trajectory, which shows the adaptability of the algorithm.

In order to be able to use a similar portion of the workspace, the first joints are a little longer to offset the short end effector. This leads to lower dexterity especially when the robot is folded. However, the algorithm is able to determine a collision-free path, but at a slightly higher computation time which in this case reaches ~21 sec.

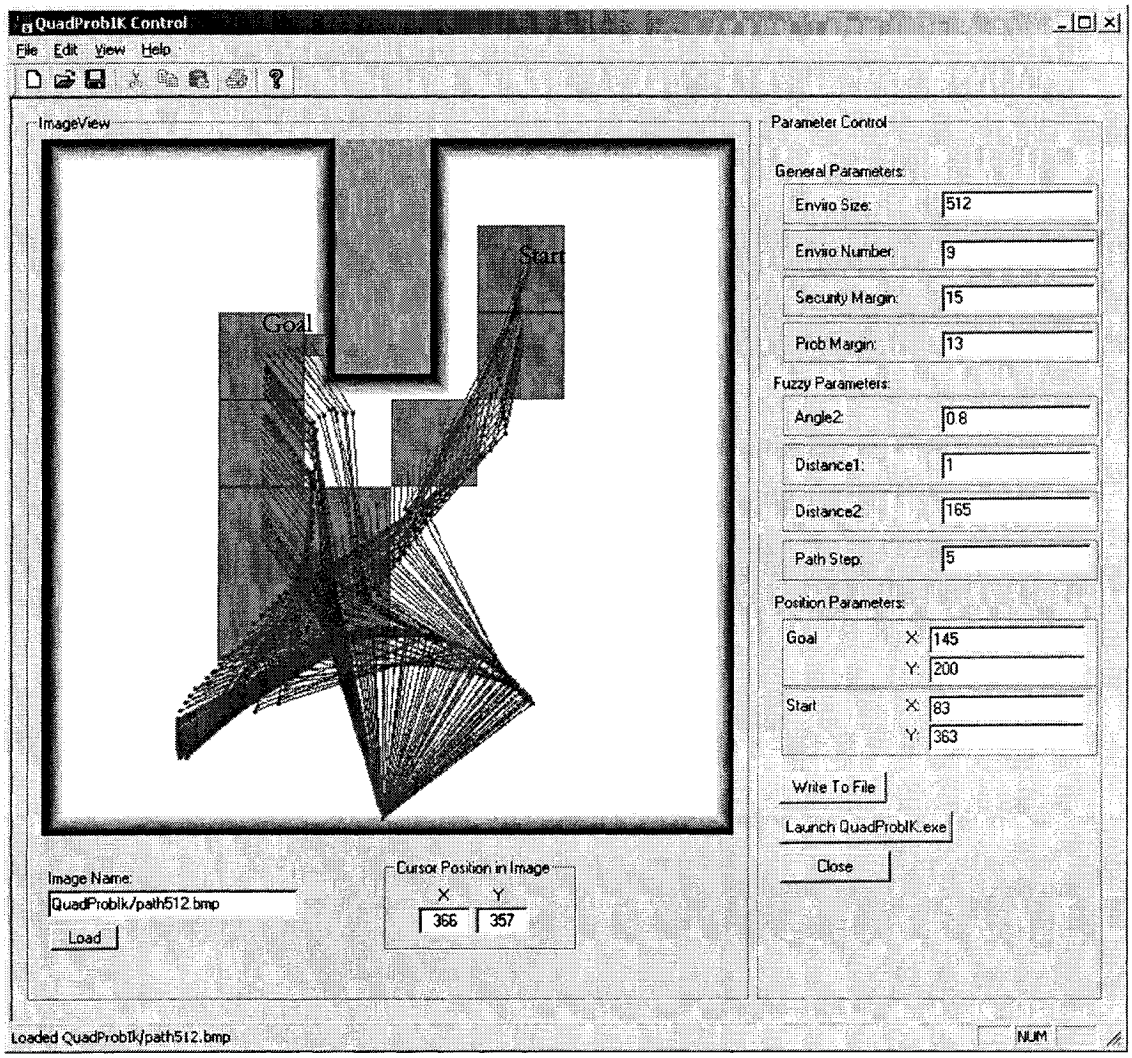


Figure 44: Path sequence for the third test case

If the target position would be higher on the left, then the algorithm would push the joints around the obstacle in order to provide a better orientation to reach the target.

In this test case, the third joint reaches the angle limit of -179° imposed, as shown in Figure 45 over the central portion of the trajectory. The algorithm succeeds to determine another configuration based on this restriction. However, a larger change in angle for all the joints is needed in the following iteration in order to bring the robot in its original series.

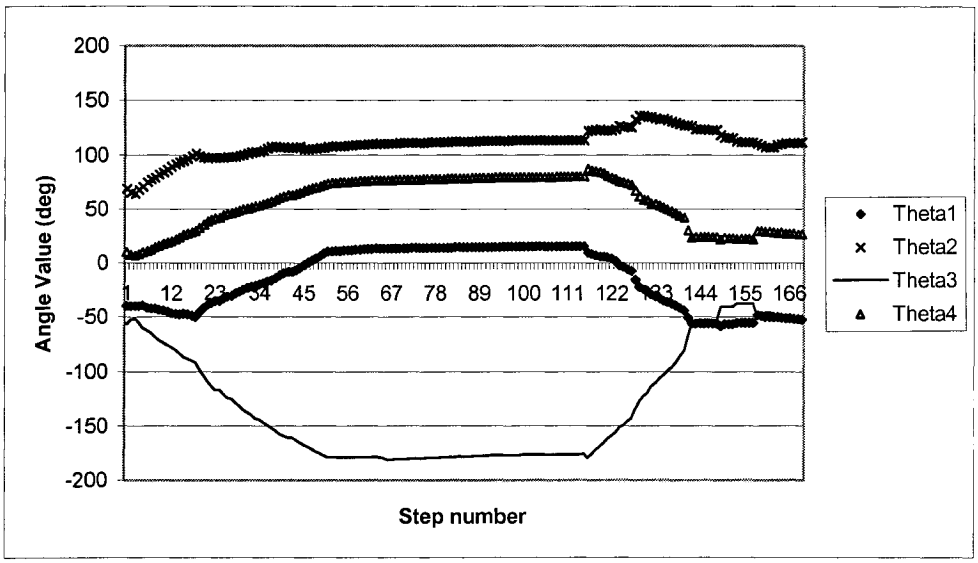


Figure 45: Angles progression for the third test case

5.4.1.4. Fourth test case

Figure 46 displays some of the robustness of the algorithm to bring the robot into a narrow corridor. Here, the robot re-orients itself to be in a better position to enter the narrow opening.

Controlling a manipulator arm in a narrow corridor has always been difficult and the focus of a considerable amount of research. In this example, the robot starts from its usual upright position and must enter a narrow corridor on its left, as shown in Figure 46.

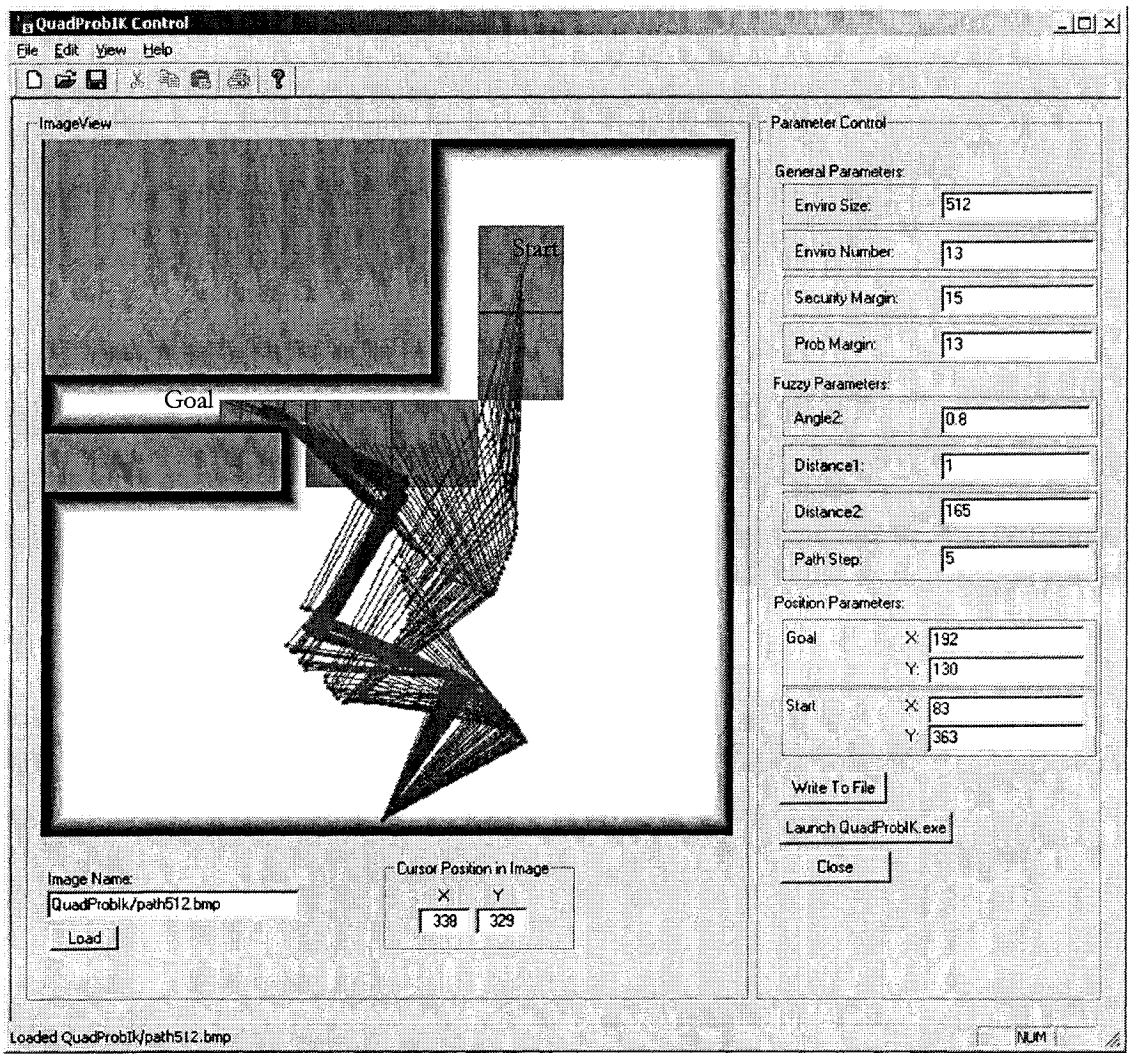


Figure 46: Path sequence for the fourth test case

The algorithm is able to determine a collision-free path and a smooth trajectory as shown in Figure 47. The angle of approach of the end effector is also re-oriented to have a better entry in the corridor. The computation and convergence rate is low in this portion of the path. It slows down once the robot enters the corridor, where the manipulator structure and nearby obstacles create contradicting forces. While the robot is able to reach the desired position, other positions further inside the opening cause the contradicting forces to be too great and the robot is not able to reach any further. At this point, the forces cause the robot to oscillate between two points, neither of which is able to help the fuzzy controller reach

the desired target. Relaxing the repulsive forces may increase the dexterity of the robot but would increase the probability of collision with the nearby obstacle.

The computation time is also slightly increased to ~26 sec because of the high resolution of the multi-resolution cells in the narrow opening. This leads to more neighbor searches that slow down the program execution.

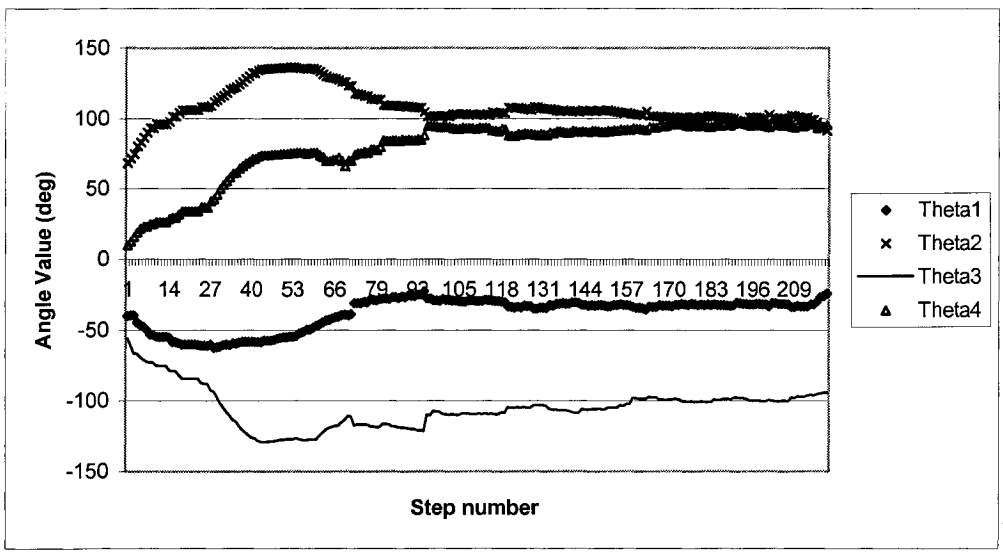


Figure 47: Angles progression for the fourth test case

5.4.1.5. Fifth test case

In the fifth test case, a 3-DOF planar manipulator is used to show that the algorithm is robust to different robot types. The lengths of the 3 remaining members are adjusted to cover a similar area to the previous examples.

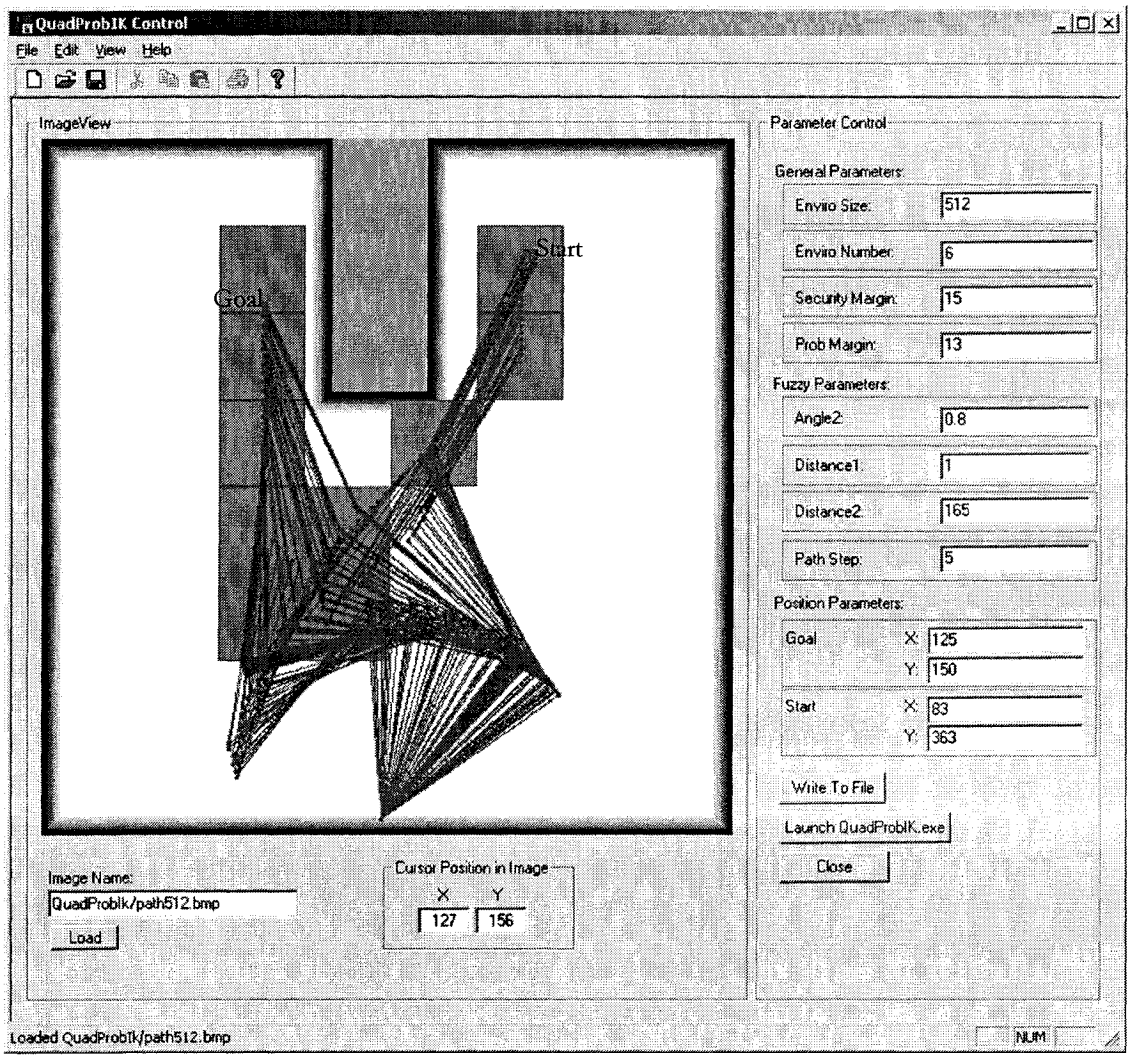


Figure 48: Path sequence for the fifth test case

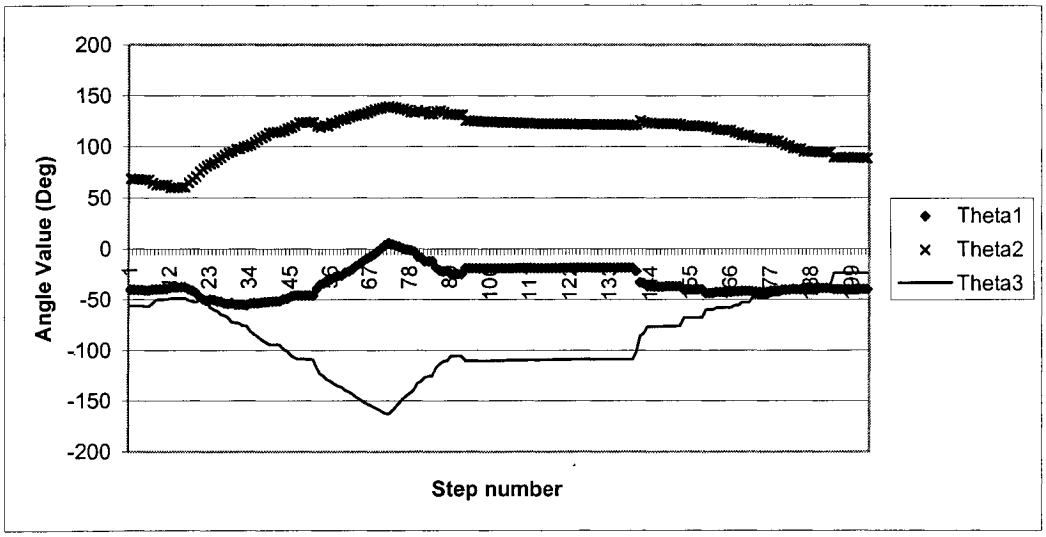


Figure 49: Angles progression for the fifth test case

Figure 48 and Figure 49 confirm that the resulting trajectory is collision-free and smooth. At first glance, the lower number of joints should reduce the computation times, but this is not the case. While a lower number of angles checks per iteration are required, the lower redundancy of the robot leads to lower convergence speeds and more iterations of the fuzzy controller. The advantage of fewer angles to compute is offset by the lower dexterity of the robot and higher number of iterations in the fuzzy controller. Nevertheless, the proposed path planning technique still works perfectly well and provides a smooth and collision-free trajectory.

5.4.2. 3D test case

In order to show the generality of the proposed approach, the path planning algorithm is extended to 3D workspaces and tested on the 6 degree-of-freedom robot defined previously in Table 9. This robot is shown in an arbitrary position in Figure 50 from two different viewpoints.

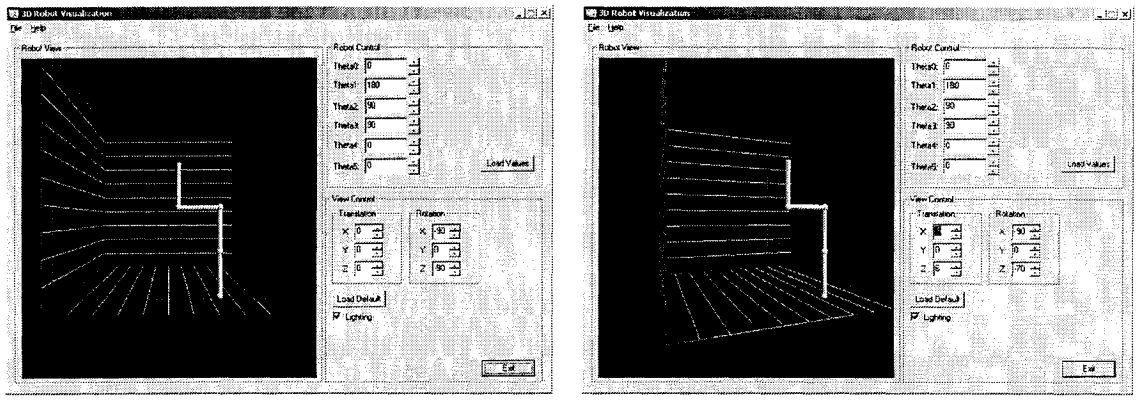


Figure 50: Sample position for 3D robot

Figure 51 represents a simple 3D environment generated from the 3D visualization tool developed for this research. It consists of a wall along the upper middle portion of the edge of an environment. This data is drawn with the program and saved as a tab-separated text file, which is then read with the main path planning program.

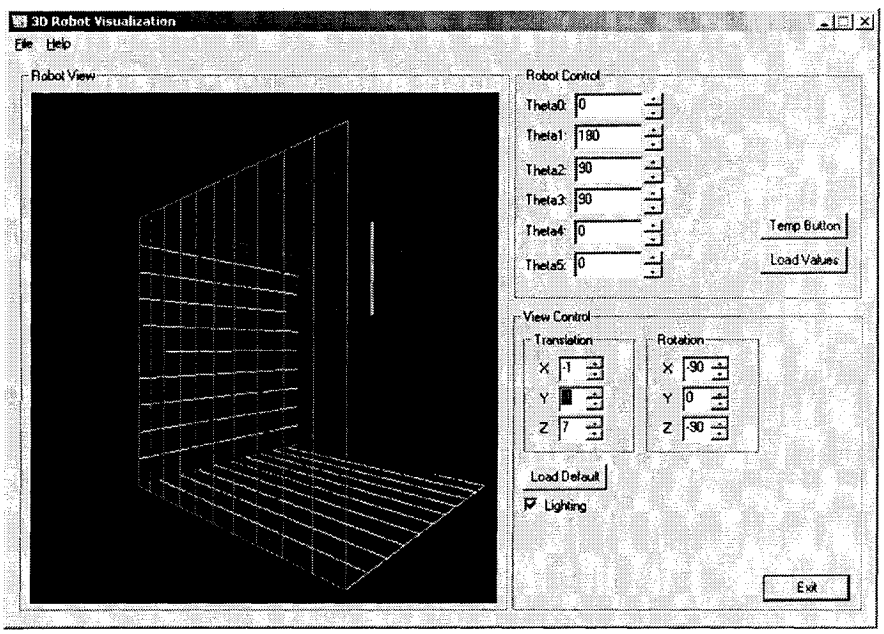
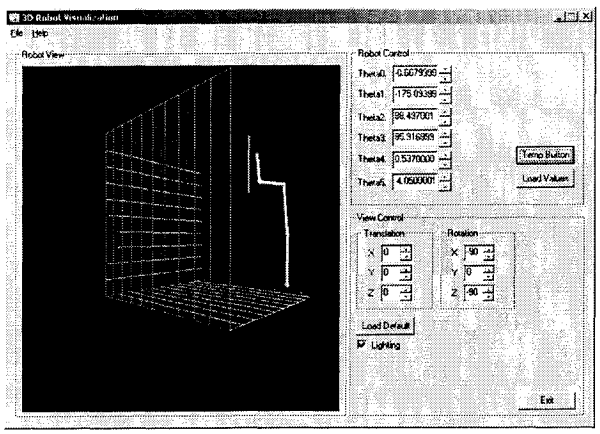


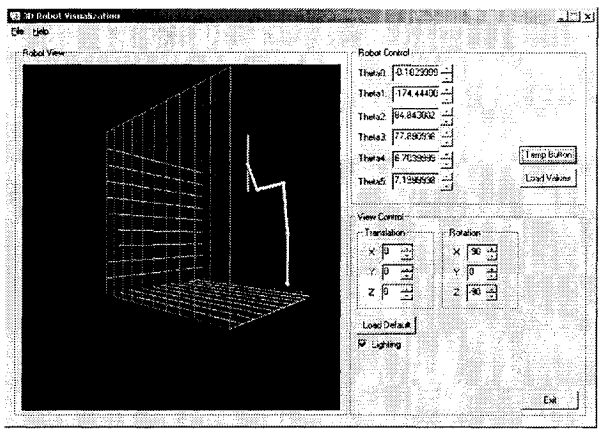
Figure 51: Obstacle in 3D space

In this test case, the robot must move around the obstacle shown in Figure 51, starting from behind it. Figure 52 displays the sequence of its path (start position is top left figure, end position is bottom right). Near the middle of the trajectory, a large multi-resolution cell

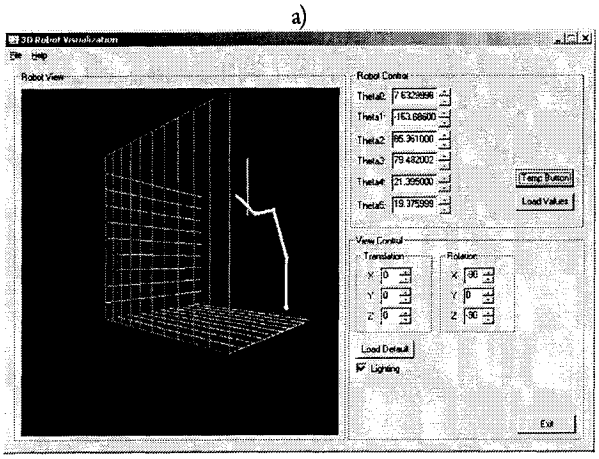
is found in the global path planning method; this brings the robot further out, but to a safe position. Close to the goal position the third joint is close to the obstacle mainly due to the conflicting attractive force of the goal and the repulsive force of the obstacle. The repulsive force coefficient could be changed in order to keep the robot further from the obstacle but here is kept as a constant that has been calibrated as discussed in section 4.3.3.1.



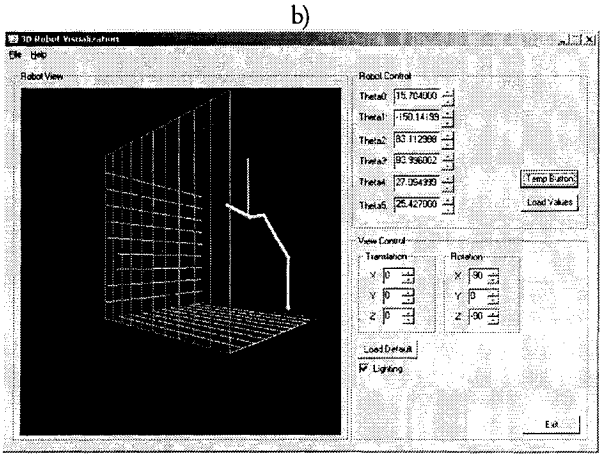
a)



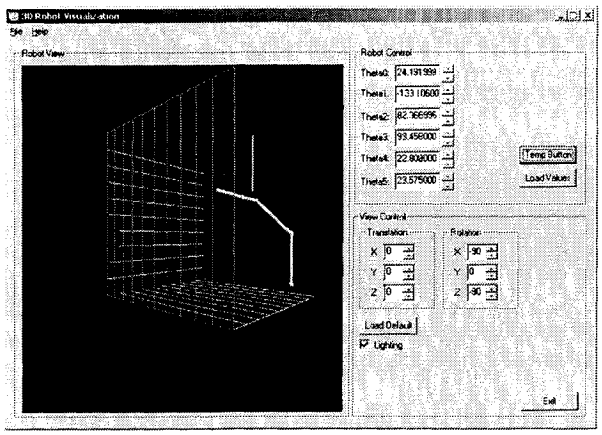
b)



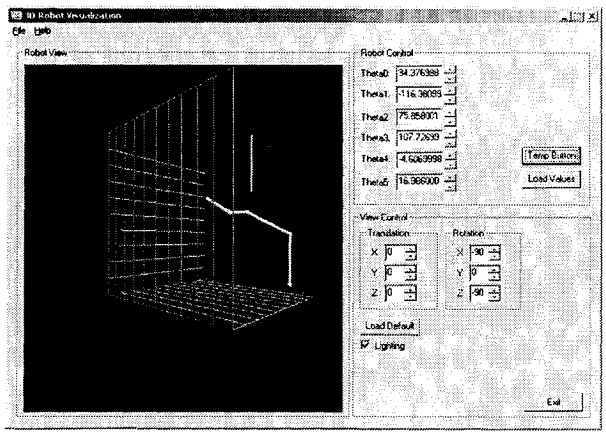
c)



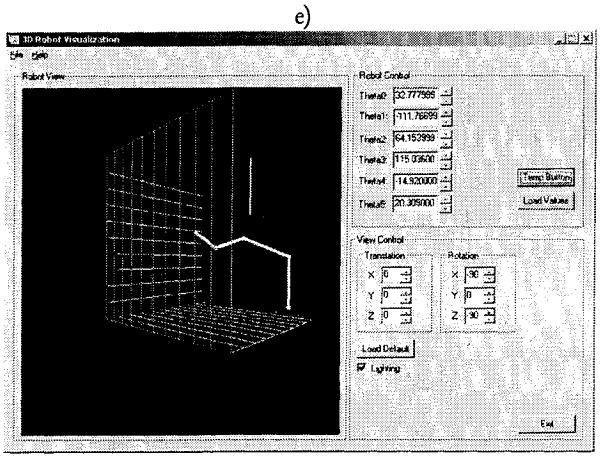
d)


3D Robot Visualization
File Help
Robot View
Robot Control
Theta0 24.191899
Theta1 -133.10600
Theta2 82.366996
Theta3 93.456000
Theta4 22.808000
Theta5 23.575000
Temp Button
Load Values
View Control
Translation
Rotation
X 0
Y 0
Z 0
X 90
Y 0
Z 90
Load Default
Lighting
Exit

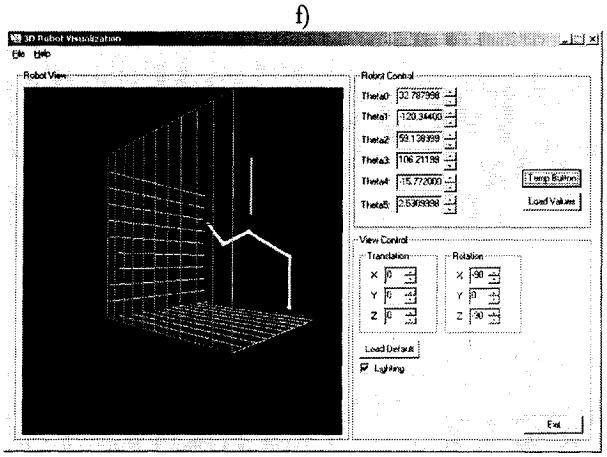
e)


3D Robot Visualization
File Help
Robot View
Robot Control
Theta0 34.376389
Theta1 -116.38089
Theta2 75.858001
Theta3 107.72639
Theta4 -4.6069998
Theta5 16.986000
Temp Button
Load Values
View Control
Translation
Rotation
X 0
Y 0
Z 0
X 90
Y 0
Z 90
Load Default
Lighting
Exit

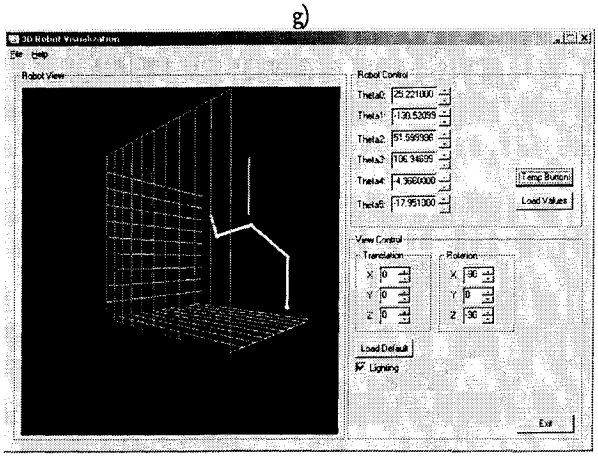
f)


3D Robot Visualization
File Help
Robot View
Robot Control
Theta0 32.777695
Theta1 -111.76899
Theta2 64.152999
Theta3 115.03600
Theta4 -14.920000
Theta5 20.309000
Temp Button
Load Values
View Control
Translation
Rotation
X 0
Y 0
Z 0
X 90
Y 0
Z 90
Load Default
Lighting
Exit

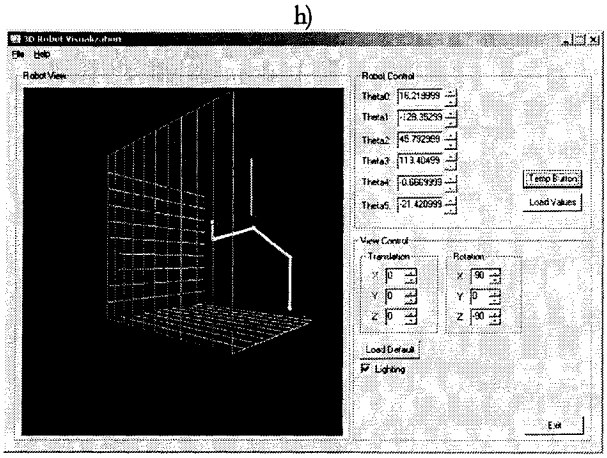
g)


3D Robot Visualization
File Help
Robot View
Robot Control
Theta0 32.787998
Theta1 -120.34400
Theta2 59.139999
Theta3 106.21199
Theta4 -15.772000
Theta5 0.8369998
Temp Button
Load Values
View Control
Translation
Rotation
X 0
Y 0
Z 0
X -90
Y 0
Z -90
Load Default
Lighting
Exit

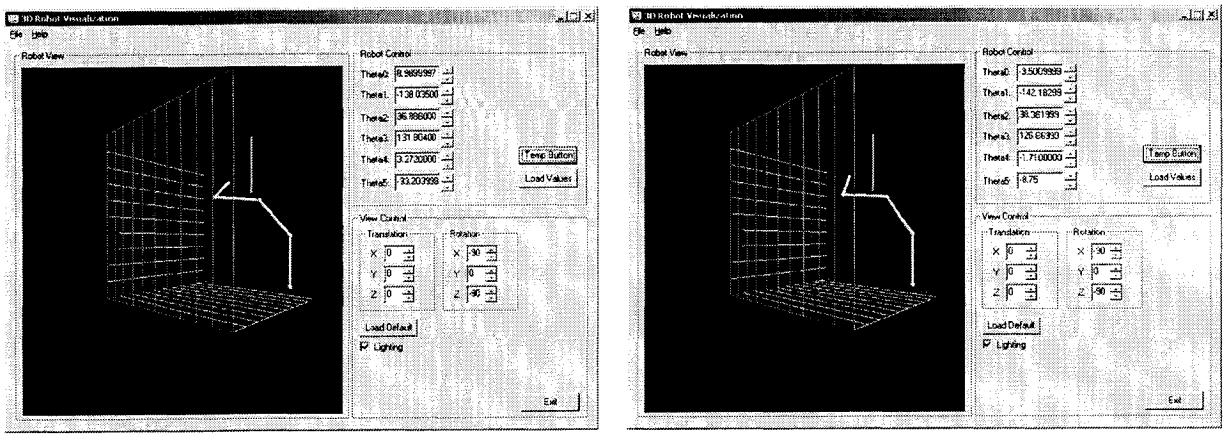
h)


3D Robot Visualization
File Help
Robot View
Robot Control
Theta0 25.221000
Theta1 -130.52099
Theta2 61.599998
Theta3 106.94695
Theta4 -4.3660000
Theta5 -17.951000
Temp Button
Load Values
View Control
Translation
Rotation
X 0
Y 0
Z 0
X 90
Y 0
Z 90
Load Default
Lighting
Exit

i)


3D Robot Visualization
File Help
Robot View
Robot Control
Theta0 16.219899
Theta1 -129.35299
Theta2 48.792969
Theta3 113.40499
Theta4 -0.6869999
Theta5 -21.420999
Temp Button
Load Values
View Control
Translation
Rotation
X 0
Y 0
Z 0
X 90
Y 0
Z -90
Load Default
Lighting
Exit

j)



k) l)

Figure 52: 3D path sequence

Despite the added complexity of 3D workspaces, computation time for this test case was approximately 17 seconds, which is comparable to 2D results mainly due to the smaller workspace size and simpler environment. However, after examining various 3D scenarios, a 50% increase in computation time for a single iteration of the fuzzy controller was found due to the added joint velocity and robot complexity.

5.5. Comparison with another approach in the literature

The proposed algorithm was tested in similar environments to that of Laliberté [14, 15] with a 4 degrees-of-freedom robot. A safe collision-free path is found using the same environment depicted in [14], despite using a simpler robot containing no prismatic joints and without the need of special heuristic methods that were implemented by the author. The fuzzy logic controller is less efficient in the early stages of the trajectory but reaches the target position very quickly afterwards, as shown in Figure 53. In addition, since Laliberté used a deterministic kinematics in velocity to model the robot, the movements of the joints are not as smooth compared to the proposed method and the kinematics solution must undergo considerable changes if a different robot is used.

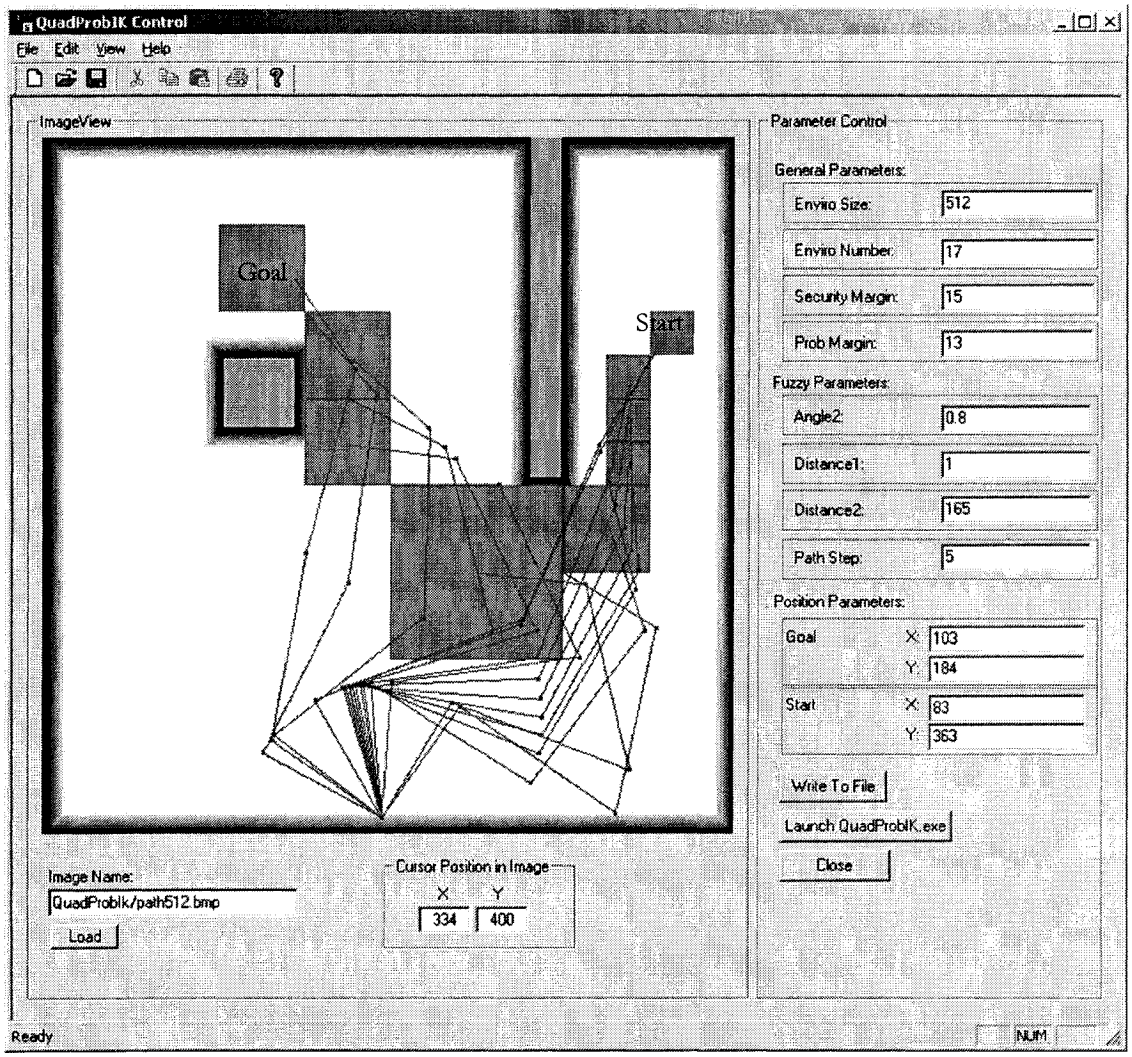


Figure 53: Output path for path planning comparison

5.6. Analysis

In the previous sections, various scenarios have been tested to demonstrate the robustness of the proposed approach and explore various situations. In order to keep the length of this chapter reasonable, only a few samples of the scenarios that have been experimented with are shown.

Through the experimentation that has been conducted on numerous scenarios, the algorithm is found to be robust for different manipulator arm configurations. The results

also demonstrated the adaptability of the algorithm to find a solution in cluttered environments at various levels of complexity.

One of the goals of this research is to provide an efficient and fast solution for the robot to track the proposed path. Table 10 displays the overall execution times (in seconds) for the test cases presented in section 5.4.1. These represent the computation time for the entire procedure: input/output of data, computation of potential fields, n-tree operations, global and local path planning phases. These execution times are fast enough for a large robot that requires slow movements as is the case in most manufacturing applications.

Case number	1	2	3	4	5
Execution time (sec)	17.040	20.230	21.370	26.920	23.460

Table 10: Execution times of test cases

Various aspects and situations influence execution time. The primary factor that increases the computation time is the complexity of the environment. The algorithm is able to converge quickly when few obstacles are found, however, when the environment becomes more cluttered, the fuzzy control requires additional iterations to converge towards a solution.

The proposed algorithm has numerous benefits, most notably its simplicity. The fuzzy logic controller does not need to be tuned for each robot; the parameters stay the same despite changes in the kinematics. The relatively short execution times allow a robotic application to follow as the program is running, allowing almost real-time control. Although different in computation demands, the approach is similar in complexity for both 2D and 3D applications. For 3D robots, the kinematics are usually much more complex, but with the proposed method, there are only minor differences, such as a third velocity to consider.

The approach does have some limitations that have been noted during experimentation. The algorithm does not perform well when an obstacle is near the base of the robot. Through testing, an empty area roughly equal to a half-circle of radius equal to the first member ensures good results. The algorithm also tends to slow down when it is folded around itself; this is particularly true for the joints nearest to the base. While the path

planning method returns a collision-free path, the computation is more intensive in those situations. The manipulator arm may be unable to reach the target position because of the location of an obstacle, for example, trying to reach behind an obstacle. The correction methods of Chapter 4 can be enough to overcome the contradicting forces. Otherwise, the algorithm is will stop if the controller is unable to determine a safe path within a pre-determined number of steps.

Other factors increase the complexity of a manipulator such as the number of degrees-of-freedom. The increase in the number of joints, however, also increases the dexterity of the manipulator and allows it to perform some tasks in an easier way. Despite the fact that more angle iterations are computed, the fuzzy controller itself was iterated less often. This is true especially in cluttered environments where high robot redundancy will allow it to overcome some problems introduced by the repulsive force exerted by obstacles. Increased dexterity helps the robot overcome situations where contradicting forces are applied to the manipulator. In our experiments, the difference in computation times between a 3-DOF and a 4-DOF robot was minimal; however, the path of the 4-DOF robot was usually smoother, changes in joint values were smaller and the system generally performed better in crowded environments as its redundancy overcame the repulsive forces constraints. 3D robots work in a similar manner, with the exception that the number of joints must be significantly higher in order to match similar redundancy.

The test cases in this section only consider redundant robots. Consequently, the increase in dexterity facilitates the movement of the robot in cluttered environments. The proposed approach can also be used with non-redundant robots to find a safe path. A lower number of joints would likely reduce the computation times in an empty environment but when used in a cluttered workspace, the robot would more often be caught in local minima as the structure is much less adaptable to find an alternative configuration.

The computation time of the algorithm is also a factor of the distance between the start and target positions. The resulting path will usually be shorter if the two points are close,

however, if one is brought closer to an obstacle, the fuzzy iterations will usually be extended. Since the path is first defined globally using multi-resolution cells, small changes in either start or target positions will yield minimal changes in overall execution times.

The use of multi-resolution cells has been found to reduce global path planning [34] by a substantial factor. In local path planning, its uses are also important. The global path will usually define a path further from obstacles since it will not follow the downslope near obstacles, as explained in section 5.4.1.2, and reduces the risk of collisions. Computation during local path planning will be shorter as repulsive forces are less important than with fixed-size cells. The risk of collision is therefore reduced with the proposed approach.

Finally, the approach is also extendable to larger robots: the repulsive forces are computed from the point in the structure where the probability of occupation is the greatest; the effects of the repulsive forces are then interpreted in the same manner.

5.7. Conclusion

In this chapter, the results of experimentation that has been conducted on numerous scenarios are shown to demonstrate the flexibility of the algorithm to deal with different manipulator arms. The results obtained demonstrate the adaptability of the algorithm to find a solution in cluttered environments at various levels of complexity.

CHAPTER 6: CONCLUSION

This work presents an approach for a path planning strategy for robot manipulators. In opposition to most techniques dedicated to mobile robots that are found in the literature, the proposed path planning approach considers the complexities of modeling a robot arm. An original solution using potential fields and fuzzy logic is presented and analyzed. The aim of the approach is to achieve a robust and adaptable algorithm that can work with various manipulator architectures and environment configurations. Two main phases are used: a global path planning phase determines the path of the end effector and a local path tracking phase configures the rest of the robot structure to make the end effector follow the pre-computed sequence of positions.

The global path planning phase uses a combination of probabilistic datasets and multi-resolution grids. The use of a probabilistic representation increases the reliability of the mapping and provides a more direct relationship to compute the repulsive potential field. An original approach to encode multi-resolution cells is proposed. This new method greatly reduces the storage requirements for storing environment information. A neighbor searching method is developed on the basis of this encoding and is applied for efficient computation of the attractive field that guides the end effector. The new approach is compared to other algorithms found in the literature and it is shown that its performance meets or exceeds that of the other techniques.

In the local path tracking phase, the complex problem of manipulator kinematics is simplified through the use of a fuzzy logic controller. With this controller, the need for determining and computing the complex equations of inverse kinematics or the use of heuristic methods is eliminated. The proposed algorithm is well suited for any manipulator architecture with minimal changes. Only the forward kinematics and its first derivatives that differ for various robot architectures need to be provided to the path planner. The approach presented by Nedungadi [22] has been improved upon for better representation of the robot kinematics, but more importantly, to consider the presence of obstacles that provide

repulsive forces acting on the manipulator structure, and extrapolated to include 3D robot setups and workspaces. The use of fuzzy logic has allowed the path planning phases to keep joint movements small between consecutive positions when traveling in a cluttered environment to reduce the probability of collisions with obstacles and keep control on singular configurations.

The proposed algorithm has been tested against different environments situations and different robot configurations. The approach has been implemented in 2D and 3D. In 2D, the computation times are small enough to allow for real-time use of the algorithm.

6.1. Main contributions of the thesis

This thesis presents original approaches to environment modelling and manipulator path planning. For global path planning, the use of probabilistic datasets and potential fields offer a simple yet robust solution. An original encoding scheme and neighbor searching algorithm are proposed to allow a compact but accurate representation of the environment that minimizes the computational effort for the search of neighbor configurations required to define a trajectory.

This classical approach is combined with an original local path tracking approach based on a fuzzy logic controller that combines the manipulator kinematics and obstacle information to determine a safe, collision-free path for the whole structure of the robot arm. The proposed setup offers the flexibility to use an arbitrary manipulator for a given environment while avoiding the extensive computation related with manipulator control based on classical inverse kinematics and has been successfully tested on various workspaces and robot setups while keeping the internal parameters constant or auto-adapted without tuning.

6.2. Future work

In order to increase the generality and efficiency of the proposed path planning solution, some areas of this research could be further explored. First of all, the approach could be

adapted to function in a dynamic environment where obstacles are in motion. This introduces great constraints on the global path planning and local path tracking phases.

Second, the original environment encoding scheme proposed in Chapter 3 provides a compact yet efficient approach to store environment information as well as the framework for fast neighbor searches. A hybrid solution could be researched that combines the simplicity of the addressing scheme proposed by [26] with the compactness and quick access to occupancy data of the approach developed in this thesis.

Moreover, although various robot architectures have been tested, they represent a subset of the different manipulators that have been constructed. It would be interesting to determine whether the proposed solution can function well in various contexts in which manipulators are used such as manufacturing and real-time applications.

Also, another goal is to integrate the algorithm with a computer vision system to generate real probabilistic data, which can be tested on a real-life robot system in order to determine difficulties that have not appeared in simulation.

Finally, future implementations of this algorithm should aim to further refine the settings of internal parameters to increase the convergence of the fuzzy controller, lower the computation times required to determine a collision-free path and provide an even more general solution.

REFERENCES

[1] S. Ando, "A fast collision-free path planning method for general robot manipulator", in *Proceedings of 2003 IEEE International Conference on Robotics and Automation*, pp. 2871-2877, Taipei, Taiwan, 2003.

[2] J. Barraquand and J.C. Latombe, "A Monte-Carlo algorithm for path planning with many degrees of freedom", in *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1712-1717, Cincinnati, OH, 1990.

[3] M.T.H. Beheshti and A.K. Tehrani, "Obstacle Avoidance for Kinematically Redundant Robots using an Adaptive Fuzzy Logic algorithm", in *Proceedings of 1999 American Control Conference*, pp. 1371-1375, San Diego, CA, 1999.

[4] S. Caselli, M. Reggiani and R. Rocchi, "Heuristic Methods for Randomized Path Planning in Potential Fields", in *Proceedings of 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pp. 426-431, Banff, AB, 2001.

[5] S. Caselli, M. Reggiani and R. Sbravati, "Parallel Path Planning with Multiple Evasion Strategies", in *Proceedings of 2002 IEEE International Conference on Robotics and Automation*, pp. 260-266, Washington, DC, 2002.

[6] H. Chang, "A New Technique To Handle Local Minimum For Imperfect Potential Field Based Motion Planning", in *Proceedings of 1996 IEEE International Conference on Robotics and Automation*, pp. 108-112, Minneapolis, MN, April 1996.

[7] L. Chengqing, M.H. Ang Jr, H. Krishnan and L.S. Yong, "Virtual Obstacle Concept for Local-minimum-recovery in Potential-field Based Navigation", in *Proceedings of 2000 IEEE International Conference on Robotics and Automation*, pp. 983-988, San Francisco, CA, April 2000.

[8] E.S. Conkur and R. Buckingham, "Increasing the maneuvering ability of highly redundant manipulators", in *Proceedings of 1997 IEEE International Conference on Robotics and Automation*, pp. 155-160, Albuquerque, NM, April 1997.

[9] A. Elfes, "Using Occupancy Grids for Mobile Robots Perception and Navigation", in *IEEE Computer*, vol. 22, no. 6, pp. 46-57, 1989.

[10] C. Ham, Z. Qu and R. Johnson, "Robust fuzzy control for robot manipulators", in *IEEE Proceedings of Control Theory and Applications*, vol. 147, no. 2, pp. 212-216, 2000.

[11] J.Y. Hwang, J.S. Kim, S.S. Lim and K.H. Park, "A Fast Path Planning by Path Graph Optimization", in *IEEE Transactions on Systems, Man and Cybernetics, Part A*, vol. 33, no. 1, pp 121-128, Jan 2003.

[12] O. Khatib, "Real-time Obstacle Avoidance for Manipulator and Mobile Robots", in *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 500-505, 1985.

[13] Y. Kitamura, T. Tanaka, F. Kishino and M. Yachida, "3-D planning in a dynamic environment using an octree and an artificial potential field", in *Proceedings of IEEE International Conference on Intelligent Robots and Systems* , vol. 2, pp.474-479, 1995.

[14] T. Laliberté, "Planification de trajectoire d'un manipulateur sériel redondant dans un environement encombré", *Master Degree Thesis*, Faculté des Sciences et de Génie, Université Laval, QC, Canada, 1994.

[15] T. Laliberté and C.M. Gosselin, "Efficient Algorithms for the Trajectory Planning of Redundant Manipulators with Obstacle Avoidance", in *Proceedings of 1994 IEEE International Conference on Robotics and Automation*, pp. 2044-2049, San Diego, CA, May 1994.

[16] S. Lee and G. Kardaras, "Collision-Free Path Planning with Neural Networks", in *Proceedings of 1997 IEEE International Conference on Robotics and Automation*, pp. 3565-3570, Albuquerque, NM, April 1997.

[17] S. Lee and G. Kardaras, "Elastic String Based Global Path Planning using Neural Networks", in *Proceedings of 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pp. 108-114, Monterey, CA, July 1997.

[18] P. Leven and S. Hutchison, "Using Manipulability to Bias Sampling During the Construction of Probabilistic Roadmaps", in *Proceedings of 2002 IEEE International Conference on Robotics and Automation*, pp. 2134-2140, Washington DC, USA, May 2002.

[19] G. Lian, Z. Sun and K. Kab II, "Smart Collision Free Motion Control for Robot Arms", in *Proceedings of 4th World Congress on Intelligent Control and Automation*, pp. 2817-2821, Shanghai, China, 2002.

[20] C.-C. Lin and J.-H. Chuang, "Potential-Based Path Planning for Robot Manipulators in 3-D Workspace", in *Proceedings of 2003 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 3353-3358, Taipei, Taiwan, 2003.

[21] Y.S. Nam, B.H. Lee and N.Y. Ko, "An Analytic Approach to Moving Obstacle Avoidance Using an Artificial Potential Field", in *Proceedings of International Conference on Intelligent Robots and Systems*, pp. 482-487, Pittsburgh, PA, Aug 1995.

[22] A. Nedungadi, "A Fuzzy Robot Controller – Hardware Implementation", in *Proceedings of 1992 IEEE International Conference on Fuzzy Systems*, pp. 1325-1331, San Diego, USA, 1992.

[23] T. Nishimura, K. Sugawara, I. Yoshihara and K. Abe, "A Motion Planning Method for a Hyper Multi-joint Manipulator using Genetic Algorithm", in *Proceedings of 1999 IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, pp. 645-650, Tokyo, Japan, October 1995.

[24] G. Oriolo, M. Ottavi and M. Vendittelli, "Probabilistic Motion Planning for Redundant Robots along Given End-Effector Paths", in *Proceedings of 2002 International Conference on Intelligent Robots and Systems*, Vol. 2, pp. 1657-1662, Lausanne, Switzerland, 2002.

[25] M.G. Park, J.H. Jeon and M.C. Lee, "Obstacle Avoidance for Mobile Robots using Artificial Potential Field Approach with Simulated Annealing", in *Proceedings of 2001 International Symposium on Industry Electronics*, vol. 3, pp. 1530-1535, Pusan, South Korea, Jun 2001.

[26] P. Payeur, "Efficient Neighbor Identification in Multiresolution kD-Trees For Mobile Robot Navigation", in *Proceedings of IASTED International Conference on Robotics and Automation*, 2001.

[27] P. Payeur, "Improving Robot Path Planning Efficiency with Probabilistic Virtual Environment Models", in *Proceedings of IEEE International Conference on Virtual*

Environments, Human-Computer Interfaces, and Measurements System", pp. 13-18, Boston, MA, 2004.

[28] P. Payeur, P. Hebert, D. Laurendeau and C.M. Gosselin, "Probabilistic octree modeling of a 3D dynamic environment", in *Proceedings of 1997 International Conference on Robotics and Applications*, 1997.

[29] M. Piaggio and A. Sgorbissa, "AI-CART: an Algorithm to Incrementally Calculate Artificial potential fields in Real-Time", in *Proceedings of 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pp. 238-243, Monterey, CA, November 1999.

[30] H. Samet, "Neighbor finding in images represented by quadtrees", in *Computer Vision, Graphics and Image Processing*, vol. 18, pp. 37-57, 1982.

[31] H. Samet, "Neighbor finding in images represented by octrees", in *Computer Vision, Graphics and Image Processing*, vol. 46, no. 3, pp. 367-386, June 1989.

[32] L. Sing, H. Stephanou and J. Wen, "Real-time Robot Motion Control with Circulatory Fields", in *Proceedings of 1996 IEEE International Conference on Robotics and Automation*, pp. 2737-2742, Minneapolis, MN, April 1996.

[33] E.J. Solteiro Pires and J. A. Tenreiro Machado, "A Trajectory Planner for Manipulators Using Genetic Algorithms", in *Proceedings of 1999 IEEE International Symposium on Assembly and Task Planning*, pp. 163-168, Porto, Portugal, 1999.

[34] M. Soucy and P. Payeur, "Robot Path Planning with Multi-resolution Probabilistic Representation: A Comparative Study", in *Proceedings of IEEE Canadian Conference On Electrical and Computer Engineering*, pp. 1127-1130, Niagara Falls, ON, May 2004.

[35] L. Tian and Z. Mao, "Fuzzy neuro controller for a two-link rigid-flexible manipulator system", in *Proceedings of 9th International conference on Neural Information Processing*, pp. 1867-1871, Singapore, 2002.

[36] Y. Wang and G.S. Chirikjian, "A New Potential Field Method for Robot Path Planning", in *Proceedings of 2000 IEEE International Conference on Robotics and Automation*, pp. 977-982, San Francisco, CA, April 2000.

[37] X. Yun and K.-C. Tan, "A Wall-Following Method for Escaping Local Minima in Potential Field Based Motion Planning", in *Proceedings of 1997 International Conference on Automation and Robotics*, pp. 421-426, Monterey, CA, July 1997.

[38] L.A. Zadeh, "Fuzzy Set", Information and Control 8, Department of Electrical Engineering and Electronics Research Laboratory, pp. 338-353, 1965.

[39] BMP image header format: *http://www.cs.umass.edu/~verts/cs32/save_24b.html*

[40] Fuzzy Logic Tutorial: *http://www.seattlerobotics.org/encoder/mar98/fuz/flindex.html*

[41] Forward Kinematics of 3D Robot: *http://prt.fernuni-hagen.de/lehre/KURSE/PRT001/course_main/node17.html*

APPENDIX A: 3D NEIGHBOR FINDING RULES

Rules to finding neighbors in 3D can be derived using the proposed multi-resolution format proposed in Chapter 3. However, they are many more possibilities as is expected by the addition of the 3rd dimension. There are now have 3 axes to consider:

1) North-South
2) East-West
3) Front-Back

In general, the neighbor finding rules depend only on the position of the cell to inspect and the direction we wish to explore. Observing how addresses are modified when a movement occurs in each of the possible directions allows us to define generic rules that can be applied repetitively over the entire map. These rules are encoded as a lookup table for efficiency. Following these rules, the neighbor location can be updated according to the start position and the information found in the lookup table, shown in Figure 56. There are two inputs to each table:

- The position of the current cell in its parent: it is represented by the column number in the table, and
- The direction of the neighbor to find: in 3D this can be one of 26 possibilities, as shown in Figure 54 where the current cell is located in the middle of the second plane, no direction being assigned to it. The numbering sequence of the 8 children for the 3D mapping is shown in Figure 55.

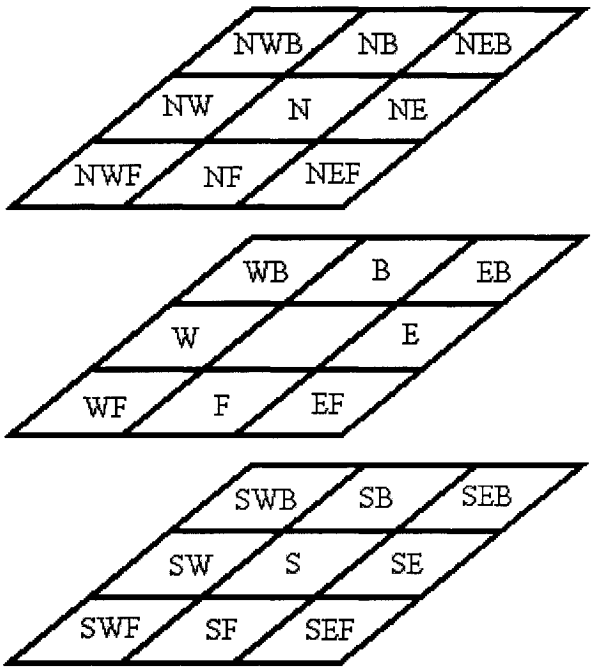


Figure 54: 26 neighbor directions in3D space

Given that the rules depend on the position of the current cell in its parent, the following numbering order is used, which is an extension of the 2D numbering scheme proposed in section 3.2.1.

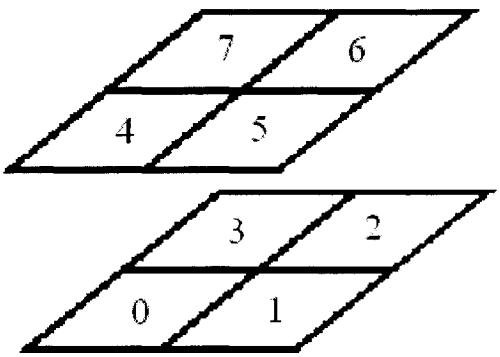


Figure 55: 3D children neighbor numbering

Figure 56 presents the rules for 3D neighbor identification.

		Position							
		0	1	2	3	4	5	6	7
Direction	NWB	W+6	+6	B+2	WB+2	NW-2	N-2	NB-6	NWB-6
	NB	+7	+5	B+3	B+1	N-1	N-3	NB-5	NB-7
	NEB	+6	E+6	EB+2	B+2	N-2	NE-2	NEB-6	NB-6
	NE	+5	E+3	E+5	+3	N-3	NE-5	NE-3	N-5
	N	+4	+4	+4	+4	N-4	N-4	N-4	N-4
	NW	W+5	+3	+5	W+3	NW-3	N-5	N-3	NW-5
	NWF	WF+6	F+6	+2	W+2	NWF-2	NF-2	N-6	NW-6
	NF	F+7	F+5	+3	+1	NF-1	NF-3	N-5	N-7
	NEF	F+6	EF+6	E+2	+2	NF-2	NEF-2	NE-6	N-6
	WB	W+2	+2	B-2	WB-2	W+2	+2	B-2	WB-2
	B	+3	+1	B-1	B-3	+3	+1	B-1	B-3
	EB	+2	E+2	EB-2	B-2	+2	E+2	EB-2	B-3
	E	+1	E-1	E+1	-1	+1	E-1	E+1	-1
	W	W+1	-1	+1	W-1	W+1	-1	+1	W-1
	WF	WF+2	F+2	-2	W-2	WF+2	F+2	-2	W-2
	F	F+3	F+1	-1	-3	F+3	F+1	-1	-3
	EF	F+2	EF+2	E-2	-2	F+2	EF+2	E-2	-2
	SWB	SW+6	S+6	SB+2	SWB+2	W-2	-2	B-6	WB-6
	SEB	S+7	S+5	SB+3	SB+1	-1	-3	B-5	B-7
	SB	S+6	SE+6	SEB+2	SB+2	-2	EF-2	EB-6	B-6
	SE	S+5	SE+3	SE+5	S+3	-3	EF-5	E-3	-5
	S	S+4	S+4	S+4	S+4	-4	-4	-4	-4
	SW	SW+5	S+3	S+5	SW+3	W-3	-5	-3	WW-5
	SWF	SWF+6	SF+6	S+2	SW+2	WF-2	F-2	-6	W-6
	SF	SF+7	SF+5	S+3	S+1	F-1	F-3	-5	-7
	SEF	SF+6	SEF+6	SE+2	S+2	F-2	EF-2	E-6	-6

Figure 56: 3D neighbor searching rules

As an extension of the strategy proposed for 2D space in multi-resolution models in Figure 21, the following figure details the rules for finding sub-neighbors. By knowing the number of the neighbor and the direction from which it was found, the children cells can be explored for uniformity. This method can be called recursively until a uniform grandchildren is found. The output value represents the location of the sub-neighbor in its parent, there can be up to 4 adjacent sub-neighbors to a cell at one level of resolution.

Direction	Sub-neighbor Location			
	Sub-neighbor 1	Sub-neighbor 2	Sub-neighbor 3	Sub-neighbor 4
NWB	1	n/a	n/a	n/a
NB	0	1	n/a	n/a
NEB	0	n/a	n/a	n/a
NE	0	3	n/a	n/a
N	0	1	2	3
NW	1	2	n/a	n/a
NWF	2	n/a	n/a	n/a
NF	2	3	n/a	n/a
NEF	3	n/a	n/a	n/a
WB	1	5	n/a	n/a
B	0	1	4	5
EB	0	4	n/a	n/a
E	0	3	4	7
W	1	2	5	6
WF	2	6	n/a	n/a
F	2	3	6	7
EF	3	7	n/a	n/a
SWB	5	n/a	n/a	n/a
SEB	4	5	n/a	n/a
SB	4	n/a	n/a	n/a
SE	4	7	n/a	n/a
S	4	5	6	7
SW	5	6	n/a	n/a
SWF	6	n/a	n/a	n/a
SF	6	7	n/a	n/a

Figure 57: Octree sub-neighbor rules

APPENDIX B: FUZZY LOGIC OVERVIEW

This section provides an overview and an example of the steps and parameters found in a fuzzy logic controller. The controller is made of three steps:

1) Fuzzification,

2) Rule evaluation, and

3) Defuzzification.

Fuzzification

In fuzzy logic, the inputs are given a more general meaning. This is done through the use of membership functions. These functions associate a weighting with each of the inputs that are processed and define function overlaps. The most common function is triangular as it reduces computation. The general form of the triangular membership functions is shown in Figure 58.

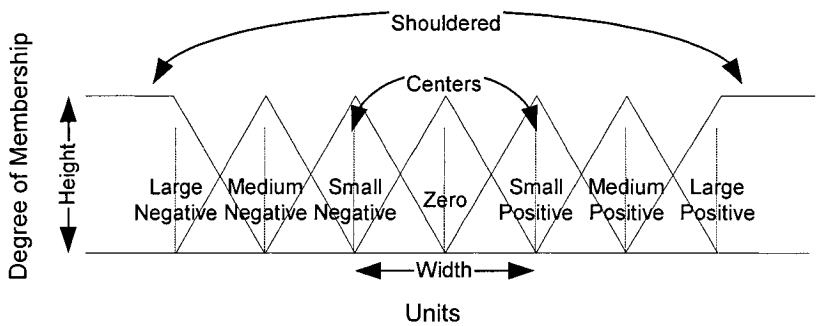


Figure 58: Membership functions

The features in Figure 58 vary depending of the application of the controller [40].

- Shape: usually triangular, although others exist.
- Height is usually normalized to 1.

- The Width of the functions varies, but follows some basic patterns: covers the entire input range, overlapping with nearby centers.
- Shouldering typically allows the functions to cover the extremes of the input range.
- Center points: determines how clustered the functions are, the center is usually assigned a membership value of 1.0, decreasing on either side.
- Overlapping allows greater flexibility in the fuzzyfication. A common value is 0.5.

Inputs are given a degree of membership (DOM) to the fuzzy functions (usually a percentage). For instance, for Figure 58, if an input is zero, than the fuzzy input will belong to the zero function at 100% and the others at 0%.

Rule evaluation

During rule evaluation, the inputs are mapped to an output using a fuzzy rule matching (also known as fuzzy associative memory). This memory bank is predefined and adapted to the application at hand. It is the heart of the fuzzy reasoning. They are based on simple IF-THEN statements that relate the inputs to the outputs, for example, IF input1 is big and input2 is small, THEN output is medium.

A good rule base will let the application converge quicker towards a solution and yield more accurate results.

Defuzzification

Defuzzification (also known as inference) will produce a tangible output value by combining the degrees of membership resulting from the rule evaluation. Numerous methods exist.

- Max-Min: selects the maximum degree of membership and assigns the output to the corresponding horizontal coordinate on the output membership distribution.

This method does not combine the effects of all the rules, but produces a continuous output and is very easy to implement.

$$output = (df_i == \max(df)) \cdot defuzz_i \quad (10)$$

Where *df* represents the membership value and *defuzz* represents the corresponding crisp value.

- Max Product (Singleton): this method does a weighted sum of the products of the degrees of membership and the magnitude of the respective function.

$$output = \frac{\sum_i df_i \cdot defuzz_i}{\sum_i df_i} \quad (11)$$

- Root-Sum-Square (Center-of-Gravity): [40] proposes a clear definition of this process as a "method that combines all the effects of all the applicable rules, scales the functions at their respective magnitudes", and computes the fuzzy centroid for each of the defuzzification functions.

$$output = \frac{\sum_i df_i \cdot defuzz\_centroid_i}{\sum_i df_i} \quad (12)$$

For completeness and ease of implementation, singletons are used.

To facilitate comprehension, a simple heating example is detailed. The two inputs are the ambient temperature and the user-set temperature. The output represents the performance required for the heating fan to operate in order to reach the desired temperature. First, the inputs are mapped to the fuzzy functions depicted in Figure 59 and Figure 60.

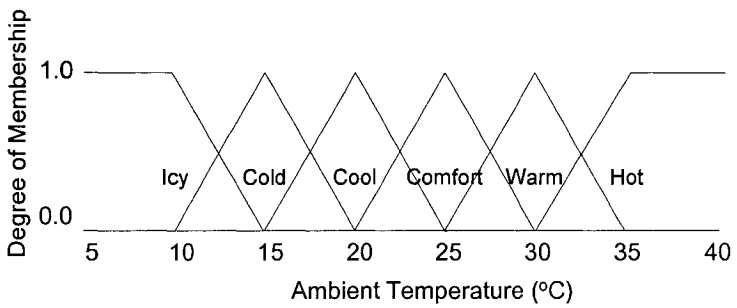


Figure 59: Sample fuzzy membership function for ambient temperature

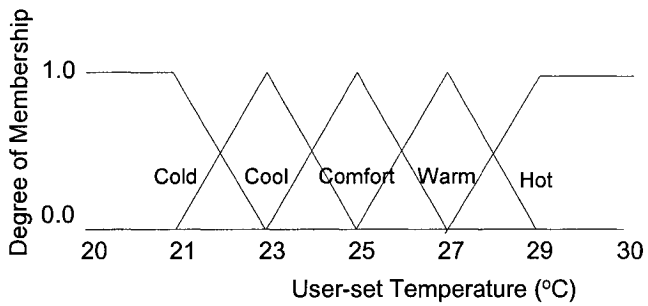


Figure 60: Sample fuzzy membership function for user-set temperature

The fuzzy associative memory is listed in Table 11.

		Ambient Temperature					
		Icy	Cold	Cool	Comfort	Warm	Hot
User-set Temperature	Hot	Max	Max	High	Low	Low	Off
	Warm	Max	High	Medium	Low	Off	Off
	Comfort	Max	Medium	Low	Off	Off	Off
	Cool	High	Low	Off	Off	Off	Off
	Cold	Medium	Off	Off	Off	Off	Off

Table 11: Sample fuzzy associative memory

Finally, the output is defuzzified by singleton functions as listed in Figure 61.

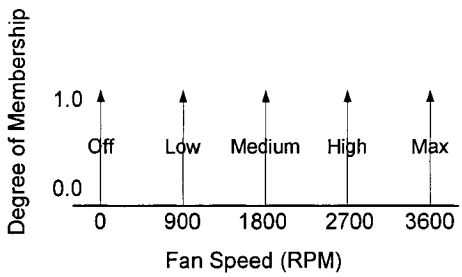


Figure 61: Sample defuzzification membership function

Using the sample fuzzy functions, let's assume that we wish to determine the output fan speed for an ambient temperature of 22.5°C and a user-set temperature of 25.5°C.

Fuzzification

The ambient temperature of 22.5 is converted to its fuzzy form where the degree of membership is 50% to Cool and 50% to Comfort. All other functions are 0.0%. The user-set temperature is 25.5 and is mapped to 75% to Comfort, 25% to Warm and 0.0% to others.

Rule evaluation

From the fuzzy rules defined in Table 11, we find the following output values:

Input		Output
Ambient	User-set	Fan Speed
Cool (50%)	Comfort (75%)	Low: min(50%,75%) = Low(50%)
Cool (50%)	Warm (25%)	Medium: min(50%,25%) =Medium(25%)
Comfort (50%)	Comfort (75%)	Off: min(50%,75%) =Off(50%)
Comfort (50%)	Warm (25%)	Low: min(50%,25%) =Low(25%)

Table 12: Sample rule evaluation

Since there are 2 values for the Low output, only the maximum is taken. Therefore, the fuzzy outputs are: Off(50%), Low(25%) and Medium(25%).

Defuzzification

A defuzzification function that can be used is the singleton method which uses the relationship detailed in Equation 11. Therefore, the output fan speed will be:

$$FanSpeed = \frac{\sum_i df_i \cdot defuzz_i}{\sum_i df_i} = \frac{df_{Off} \cdot defuzz_{Off} + df_{Low} \cdot defuzz_{Low} + df_{Medium} \cdot defuzz_{Medium} + df_{High} \cdot defuzz_{High} + df_{Maximum} \cdot defuzz_{Maximum}}{df_{Off} + df_{Low} + df_{Medium} + df_{High} + df_{Max}} \quad (13)$$

$$FanSpeed = \frac{0.5 \cdot 0 + 0.25 \cdot 900 + 0.25 \cdot 1800 + 0.0 \cdot 2700 + 0.0 \cdot 3600}{0.5 + 0.25 + 0.25 + 0.0 + 0.0} = \frac{0 + 225 + 450}{1} = 675 rpm$$

The fan speed will therefore be 675 rpm in order to reach the user-set temperature of 25.5°C from the ambient temperature of 22.5°C.

APPENDIX C: BMP FILE FORMAT

This project makes use of BMP images extensively to encode the environment and to validate results. It provides an effective way to display the obtained results. The format of the BMP file is simpler than most image formats (such as JPG or GIF) as it does not make any compression of data and can be easily encoded. This appendix explains the header and file format of the BMP file and is inspired from [39].

The file format consists of three concatenated parts as shown in Figure 62:

1) The BMP header where general BMP parameters are stored;
2) The File header containing specific file and image information; and
3) The Image data.

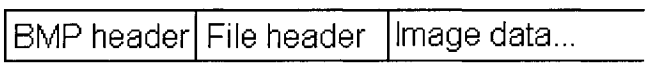


Figure 62: BMP file format

1) The BMP header consists of the following:

Parameter name	Size	Description
File signature	2 Bytes	Must be « BM », bitmap identifier
File size	4 Bytes	Not used, from an earlier BMP version, can be 0
File reserved	4 Bytes	Always 0
File Offset	4 Bytes	Size of header, or offset to image data, usually 54 (is different when using specific color pallets)

Table 13: BMP header format

2) The file header follows this format:

Parameter name	Size	Description
Image header size	4 Bytes	Size of file header (always 40)
Image width	4 Bytes	Width of image in pixels
Image height	4 Bytes	Height of image in pixels
Image Num Plane	2 Bytes	Number of planes in image, usually 1
Image Num Bits Pel	2 Bytes	Number of bits per pixel, can be 1, 4, 8, 24

Image Compression	4 Bytes	Compression used: 0=no compression, 1=RLE8...
Image size	4 Bytes	Size of image (in number of bytes) with padding[2]
Image Hor Res	4 Bytes	Horizontal resolution, not used
Image Ver Res	4 Bytes	Vertical resolution, not used
Image Num Color	4 Bytes	Number of colors, if different from Image Num Bits Pel
Image Num Imp Colors	4 Bytes	Number of important colors, usually 0

Table 14: BMP file header format

Although not very common, the standard allows the use of a color pallet.

3) Image data:

The image data is then included sequentially to the end of the headers. They are written bottom-up, left-right (therefore, the lower left pixel is the first to be written). There are no footers to indicate the end of the file. The image data of the first pixel is placed first, which contains 1,4,8 or 24 bits, then the following pixel. This is repeated until the entire image has been covered.

[2] For 24-bit/pixel images, $imageSize = imHeight * 3 * (imWidth + (4 - imWidth \bmod 4) * (imWidth \bmod 4 == 0))$

APPENDIX D: VISUALIZATION TOOLS OVERVIEW

This section details the 2D and 3D visualization tools developed to test the proposed method.

In 2D, Figure 63 illustrates the default status of the application without loading any image files. The implementation contains 2 different parts: a picture placeholder to load the images generated by the path planning methods. The second part contains fuzzy logic and other parameters as well as relevant information that can be changed prior to execution.

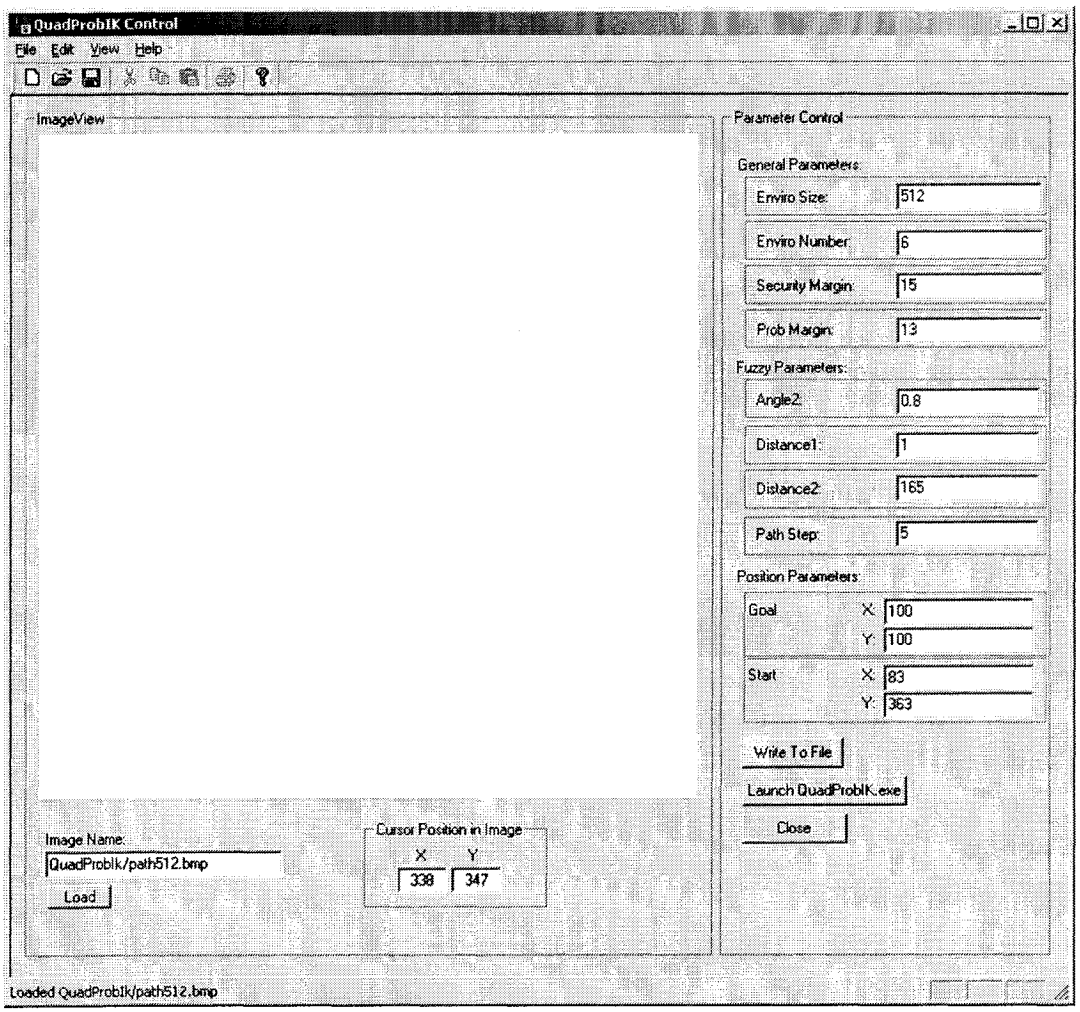


Figure 63: Sample screenshot of 2D visualization program

The following parameters can be changed:

1) General Parameters:
    a. Enviro Size: Size of the environment.
    b. Enviro Number: Identifier of environment used.
    c. Security Margin: Number of cells with uncertain status around the obstacles.
    d. Prob Margin: Threshold on probabilistic data used for global path planning.
2) Fuzzy Parameters:
    a. Angle2: Distance between defuzzification functions (output).
    b. Distance1: Distance between fuzzy functions of end effector displacement (input1).
    c. Distance2: Distance between fuzzy functions of end effector velocity (input2).
    d. Path Step: Distance between intermediate steps in number of cells.
3) Position Parameters:
    a. Goal X and Goal Y: Cartesian coordinates of goal position in cells.
    b. Start X and Start Y: Cartesian coordinates of start position in cells.
4) Cursor Position in Image: Displays position of cursor over image.
5) Image Name: Image to load and display in application.

In 3D, the implementation makes use of OpenGL libraries to display the environment and robot information. The viewing area can be rotated and translated along all three axes to provide better visualization. Also, joint values for the robot can be changed and updated in the viewing area. The 3D implementation is displayed in Figure 64.

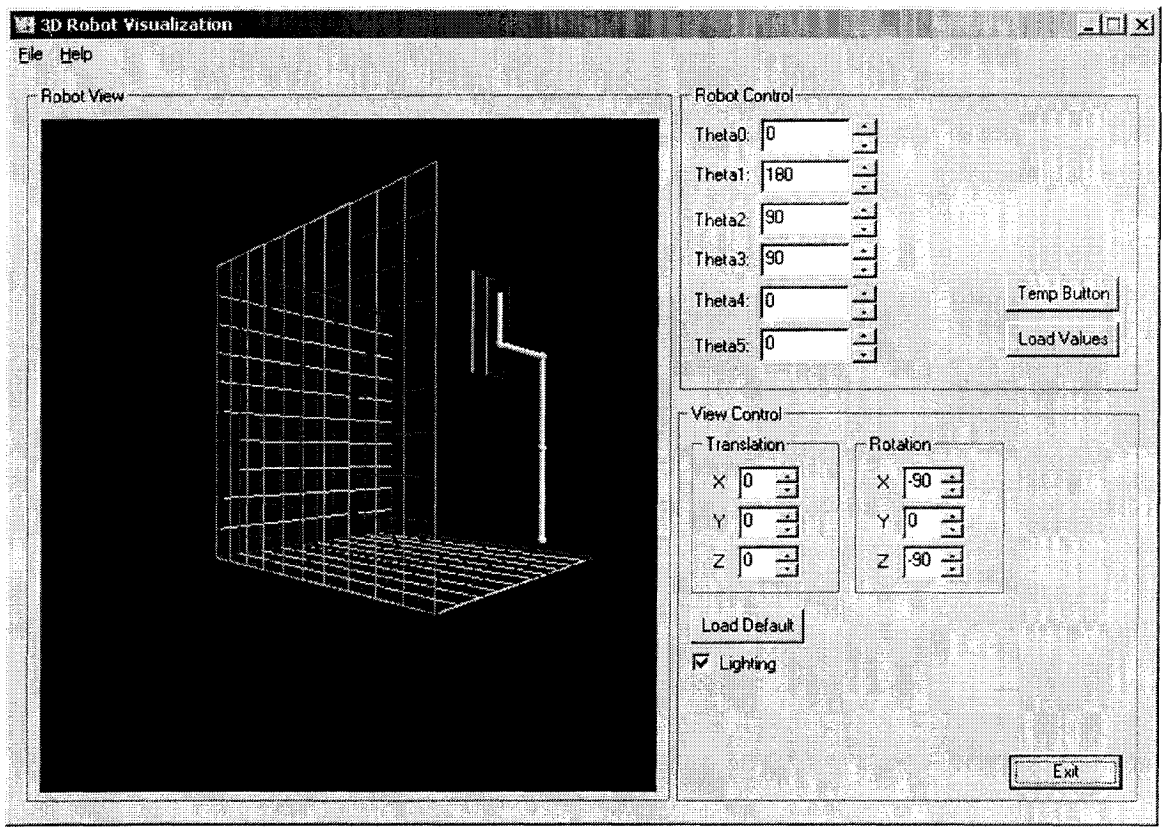


Figure 64: Sample screenshot of 3D visualization program