CONTROL OF AN OBSTACLE AVOIDER MOBILE ROBOT USING IMAGE PROCESSING

by

Turgay Kulgu

B.S., Mechanical Engineering, Istanbul Technical University, 2003

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Mechanical Engineering
Boğaziçi University

2007

ACKNOWLEDGEMENTS

I wish to thank to Assoc. Prof. Dr. Eşref Eşkinat for his invaluable guidance and help during preparation this dissertation. I want to mention his patience, giving me hope even in the dead-ends.

I also would like to thank to İshak Mizrahi for his help. Besides I am grateful to my family for their support during this study.

ABSTRACT

CONTROL OF AN OBSTACLE AVOIDER MOBILE ROBOT USING IMAGE PROCESSING

This study investigates control of a mobile Lego robot as it moves towards the specified target without hitting the obstacles with the help of image processing. An experimental set-up was designed and experiments were conducted.

In addition, most common path planning algorithms were examined and Matlab was used to make simulations of these algorithms in this study. In the light of comparison of the simulation results, VisBug Algorithm was selected as the most suitable algorithm for our study. Finally, necessary control commands to control the Lego robot were written in Microsoft Visual C++ and, are implemented in a user interface.

ÖZET

ENGELLERDEN KAÇAN MOBİL ROBOTUN GÖRÜNTÜ İŞLEME İLE KONTROLÜ

Bu çalışma, görüntü işleme yardımıyla mobil bir robotun ortamda bulunan engellere çarpmadan ilerlemesi, belirlenmiş olan hedefe doğru yol alması için gerekli rotanın çizilmesi ve gerekli kontrol komutlarını sağlayan yazılımın Microsoft Visual C++ yardımıyla oluşturulması ile ilgilidir. Bu amaçla bir deney düzeneği tasarlanmış ve oluşturulmuştur.

Bu çalışmada ayrıca, yol planlama ile ilgili mevcut algoritmalar incelenmiş, incelenen bu algoritmalarla ilgili Matlab'te simulasyonlar yapılmıştır. Bu simulasyonların sonuçlarının karşılaştırılmasıyla, VisBug algoritmasının bu çalışmada kullanılmaya en elverişli yöntem olduğu sonucuna varılmıştır. Son olarak, robotun kontrolü için gerekli komutlar Microsoft Visual C++ programında derlenmiş ve yine Visual C++ yardımıyla oluşturulmuş arayüze taşınmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

α	The robot's direction is in degrees
β	The target's direction is in degrees
γ	The movement direction of the robot is in degrees
C	Configuration space
HSV	Hue, saturation, value
IR	Infrared
LCD	Liquide crystal display
MFC	Microsoft foundation class
MHz	Megahertz
NQC	Not Quite C
PLC	Programmable logic controllers
PRM	Probabilistic Roadmap
RCX	Robotic command explorer
RGB	Red-Green-Blue
ROM	Read only memory
RRT	Rapidly Exploring Random Tree
SRAM	Static random access memory

1. INTRODUCTION

Robotics is a special engineering science dealing with designing, modeling, controlling and robots' utilization. Nowadays robots accompany people in everyday life and take over their daily routine procedures. The range of robots' utilization is very wide, from toys through office and industrial robots finally to very sophisticated ones needed for space exploration [1].

For many years, scientists have experimented with providing machines "artificial intelligence," or computer brains. Nowadays, robots are used for both work and entertainment, and they will be useful for many more things in the near future.

Robot motion planning is defined as "A Journey of Robots, Molecules, Digital Actors, and other Artifacts" in a survey paper by a researcher, J.-C. Latombe, who has influenced the field significantly during the last decades [2]. The wide applicability of motion planning algorithms and the progress which has been made in the field are summarized by this description.

To sum up in note, robot motion planning tries to solve the task of finding a collision-free path for a rigid robot through clearly defined static rigid objects or obstacles. This basic task becomes computationally hard to solve as soon as the number of degrees of freedom increases above a few. Nowadays, motion planners are deal with solving complex practical problems for robots with many degrees of freedom operating in cluttered dynamic environments. The complexity of the task has been attacked by means of powerful heuristics and specifically randomized algorithms. Also, the research field has been enlarged to include a number of sub domains. Diversification of the basic problem include algorithms which consider moving obstacles and operate under kinematic and dynamic constraints limiting robot movements. While some of the planners dealing with complex structures of the robot including articulated mechanisms or deformable geometries, or robots which are capable of reconfiguring their modular structure during the planning process, other planners emphasize on computing optimal

trajectories given some performance measure. With the help of dedicated algorithms or motion plans computed based on incomplete and error-afflicted sensor information, multiple robots can be coordinated [3].

Navigation on rough terrain, as encountered in planetary missions or off-road settings outdoors are in some motion planners' interest. Such environments pose a number of challenges not encountered in classical motion planning.

Increasingly, many of the above mentioned issues are taken into account by the planners. Planners have successfully applied mobile robots to real-world systems for non-trivial tasks where robust behavior is crucial. Therefore, mobile robots are also becoming increasingly practical.

The following section is not intended to be a complete summary of all robot motion planning research. On the other hand, it gives an outline of the development in the field from the beginnings to the nowadays' topics of interest. Various algorithms presented here are drawn from rough-terrain planning.

There is an ambiguity in literature about the terms motion planning and path planning. In some terminologies, motion planning refers to the extensions of the purely geometric path planning task. The extensions include the computation of collision-free paths under dynamic constraints, using sensor information, cooperative path planning etc.. In other usage, the term motion planning refers to the basic task whereas "trajectory planning" describes the extended variants. In this study "motion planning" and "path planning" are used synonymously and the distinction made explicit where relevant. Similarly, the terms "path" and "trajectory" are used interchangeably where not explicitly stated otherwise.

To be able to control robots, PLC drivers (Programmable Logic Controllers) or other specialized regulators are used. The aim of this thesis is to realize more flexible control strategy of a mobile robot model using a PC computer. In this study, Microsoft Visual C++ programming language was used.

Since, testing, controlling of the industrial type of robot would be very expensive, using the Lego type mobile robot is more logical because it is quite cheap and despite the fact that it is simple and it has limited possibilities, it allows to present and understand the idea of controlling mobile robots.

Before starting to this project, we assumed that, this work would provide and demonstrate a robot that avoids randomly placed obstacles on the floor and grabs the images via a web cam and analyzes the images with Matlab to find a randomly placed particular object. The robot would be built by Lego bricks with an onboard camera.

On the other hand, all of seven path planning algorithms examined in this study need a pre-determined map of obstacles, an initial point and a goal point to calculate the shortest path. Therefore, we changed the idea of using an onboard camera and decided to use a dome type camera which sees all workspace from bird-eye view. Because, path planning algorithms that will be mentioned in the following chapters do not have the ability to look ahead for unidentified obstacles. In addition to these, it is not possible to calculate the shortest path with an onboard camera.

After beginning to study on this subject with an extended literature survey we have realized that, although Matlab is an excellent programming language, use of Matlab is not so common in real time image processing applications, because, Matlab is not fast enough to accomplish real time image processing task. Then, it has been decided to write the program codes using Microsoft Visual C++.

One of the most important aspects of this project is "path planning". After deciding on the programming language to be used, path planning algorithms were started to be examined. If we design a mobile autonomous vehicle, path planning would be one of the basic requirements. The arbitrary initial position of the vehicle is called "A" and desired goal position called "B". In order to accomplish this task, several methods that will get the autonomous vehicle from A to B can be used.

While executing its trajectory, the vehicle must not come into collision with

obstacles that are randomly distributed on the workspace. Indeed, the combination of path-planning, obstacle avoidance and uncertainty of the position of the obstacles make the project rather complicated.

There are seven most common approaches which are applied in path planning problems;

1. Voronoi Approach
2. Bug Approach
3. Visibility Graph Approach
4. The Cell Decomposition Approach
5. The Potential Field Approach
6. Probabilistic Roadmap
7. Rapidly Exploring Random Tree

While generating the configuration space for the robot motion planning, these methods use the Minkowski-Sum approach. Each of these seven most common methods was particularly examined in this study. Their simulation diagrams except Probabilistic Roadmap and Rapidly Exploring Random Tree, were made via Matlab and can be seen in the following sections.

2. LITERATURE REVIEW

Many researchers dealt with controlling mobile robots in recent years. Altaira is one of the visual languages which are intended for controlling small Lego robots. Altaira combines the data from sensors, navigation, and the current state to determine actions to be taken by the robot and changes required in the system state. It is a rule-based language [4].

In the late 1960s, first robot motion planning algorithms emerged; the visibility graph technique is using a graph search to find the shortest path past polygonal obstacles for a robot represented by a point can be considered as an example [5].

Cacoa, that is created by Heger, Smith and and Cypher, is one of the visual programming language for children. It is also attended the Visual Programming Challenge '97. The challenge is about navigating a robot over an arbitrary track of Lego street tiles, and creating a map of the entire layout. Cocoa both provided a solution and won the competition [6].

Techniques developed in the fields of evolutionary computation, adaptive systems, agents, and artificial neural networks can be used in entertainment robotics in order to provide easy access to the robot technology. A number of user-guided approaches based on the techniques from these research fields have been developed. These techniques include user-guided behavior-based systems, user-guided evolutionary robotics, user-guided co-evolutionary robotics, and morphological development. All these techniques are applied to allow children to develop their own robot behaviors in a very easy and fast manner. In his studies, Lund shows examples with development of Khepera robots and LEGO MINDSTORMS robots, including the World Cup'98 stadium, the Co-evolutionary Robot Soccer Show, the Toybot Soccer Player, the Lego Interactive Football, and RoboCup Junior Rescue [7].

The idea of reducing a robot to a point was formalized in 1979 [8]. This knowledge caused the improvement of the key concept of configuration space [9]. The configuration space is a parameter space which represents all degrees of freedom of a robot and it is denoted by C. The robot can be represented as a single point in such a space; the configuration of the robot is defined by its coordinates. Obstacles in the workspace of the robot are translated into obstacles in configuration space. Hereafter, the motion planning task is reduced to finding a path through the subspace of C, which is not occupied by configuration space obstacles, is called free space.

A complete path planner is defined as an algorithm which always finds a solution to the passed path planning problem whether one solution exists or not exists. Leading to the development of heuristic path planners, such path planners have proven to be computationally forbidding for nontrivial tasks.

One of the methods in path planning algorithms is called cell decomposition algorithm. This type of algorithm relies on decomposing the free space into "cells", such that a path between any two configurations in a cell can easily be generated. A connectivity graph between adjacent cells is constructed and searched. In the approximate cell decomposition method, cells of some predefined shape (e.g. rectangloids) are used in a way that the union of all cells is strictly included in free space [10]. On the other hand, exact cell decomposition algorithms subdivide the entire free space into cells of different geometries, for instance; trapezoids for polygonal obstacles. Cells and not individual configurations are linked by the connectivity graph. The advantage here is that, some freedom remains in the found channel (sequence of cells to traverse) to compute a path for instance; accommodates dynamic constraints. Depending on the discretisation resolution, the algorithm is complete. Hence, approximate cell decomposition is resolution-complete, it means; depending on the discretisation resolution the algorithm is complete, while exact cell decomposition is a complete algorithm.

One of the other heuristic approaches to path planning makes use of artificial potential fields [11]. Despite the fact that potential field methods were originally designed for on-line collision avoidance, by means of combining them with graph searching

techniques, these methods can be extended to perform global motion planning tasks [12]. In potential field methods, while an artificial repulsive potential is generated by the obstacles, an attractive potential is generated by the goal configuration. In order to compute artificial forces acting on the robot the superposition of both potentials is used. At every configuration the resulting force is considered the most promising direction of motion.

Since only the neighborhood of the current configuration is considered when computing the forces, potential field algorithms are considered as local methods. That is why, the potential field approach is well-suited to operate in unknown or uncertain environments using sensor data. Compared to the other methods, potential field approaches can be very efficient but they can get trapped in local minima since they use steepest descent optimization. In order to get rid of local minima problem, either potential functions without local minima (navigation functions) have to be designed or mechanisms to escape from this problem have to be used.

In order to avoid local minima, one successful approach was introduced with randomized potential field planners (RPP) [12]. When a local minima is detected before proceeding with the steepest descent, a series of random motions are performed by RPP algorithms.

Building a probabilistic roadmap (PRM) is formed by another randomized path planning approach [13]. Using a local planner to build the roadmap, the PRM algorithm connects random samples in the configuration space. Before being introduced into the data structure, every sample and local path needs to be checked for collisions with obstacles. The usage of collision checker has the advantage of avoiding the costly explicit representation of free space. Including high numbers of degrees of freedom, these have been successfully applied to a wide range of problems by PRM planners.

The category of sampling-based motion planners has not only RPP but also PRM. The obstacles are not explicitly represented by sampling-based algorithms but these algorithms rely on being able to determine for any desired configuration whether it lies

in an obstacle or in free space. Instead of this binary criterion, sometimes the distance to the nearest obstacle is computed.

2.1. Kinematic Constraints

Some kinematic constraints can apply to the path planning scenario which influence the way in which a motion plan can be computed. A short categorization of kinematic constraints is given below together with the required modifications of motion planning techniques.

Holonomic constraints are defined as the constraints on the parameters composing the configuration space of the robot. In addition to this, holonomic equality constraints are defined as the equality relations among the parameters which can be solved for one of them. Hence, that parameter can be eliminated and as a result of this a subspace of C used becomes one dimension less. Multi-independent holonomic equivalence constraints can apply which reduce the dimensionality of the resulting configuration space accordingly. Moreover, time-dependent constraints can occur. Path planning under the influence of holonomic equality constraints is fundamentally the equal problem as without, albeit in a different configuration space. For instance; holonomic equality constraints arise from articulations in the robot structure [14].

Holonomic inequality constraints are defined as the inequality relations between the parameters of the configuration space. While C has the original dimensionality, these constraints define a subspace of C. These constraints typically correspond to obstacles or e.g. mechanical stops.

The configuration space includes both the parameters and their time derivatives in a way that the time derivatives cannot be eliminated and the non-holonomic constraints are constraints on the parameters composing the configuration space. The dimensionality of the configuration space is not reduced by non-holonomic constraints but the dimension of the space of possible differential motions at any configuration can be reduced. Well known case for non-holonomic equality constraints form from car-like

steering mechanisms without sliding effects. The vehicle's configuration space on a plane is three-dimensional: the orientation and a two dimensional Cartesian position. The midpoint's velocity between the non-steering rear wheels is always tangent to the vehicle orientation. Thus, one degree of freedom from the space of achievable velocities is eliminated.

Without reducing its dimensionality, the set of achievable velocities of the robot is usually restricted by non-holonomic inequality constraints. For instance, mechanical stops limit the steering angle of the front wheels in the car-like steering example. As a result, the space of possible velocities at any configuration is further restricted to a 2-D cone around the neutral position.

Non-holonomic motion planning considers systems where non-holonomic constraints apply to the robot. The solution path has to be collision free and it must consist of motions which are applicable by the robot. These kinds of paths are called feasible paths.

Considering the locally controllable robots, it is argued that, the existence of a free path is equivalent to the existence of a feasible path because of the fact that the free path can be approximated by an arbitrarily close feasible path [14]. The basis of a family of algorithms which decompose the computation of a feasible path into two phases is formed by this property. First of all, a free path is computed for a holonomic robot geometrically equivalent to the non-holonomic one studied and then, a feasible path is derived considering the non-holonomic constraints, knowing that such a path is existent [15].

In addition to these, probabilistic roadmaps have also been applied to motion planning with nonholonomic constraints [16]. The focus of attention lies on the local planner which must enforce the non-holonomic constraints. Moreover, another important thing is the computational complexity of the local planner since it is invoked not only for computing a single-shot path but a graph covering the entire configuration space.

Moreover, before multiple queries can be efficiently retrieved, this roadmap has been computed. The two-level approach which is mentioned above has also been applied to non-holonomic PRM planners. Furthermore, it has been extended to comprise multiple levels of refinement [17].

2.2. Kinodynamic Motion Planning

Kinodynamic motion planning is related with the situations that both kinematic constraints and dynamic constraints (such as bounds on velocities, accelerations, forces and torques) are considered [18].

Generally, motion planning has often been decoupled by means of the control engineering task of following the computed trajectory satisfying system dynamics. While a purely kinematic free path is computed in the first phase of such two-stage approaches, in the second stage this purely kinematic free path is converted to a trajectory. The trajectory is a time-dependent path which can be executed by the dynamic system.

2.3. Programming Environments for the Robotic Command Explorer

The following list gives an overview of the various programming environments for the Robotic Command Center which is denoted by RCX. In addition to this, Table 2.1 includes a quick reference for these programming environments.

2.3.1. Graphical Languages

This category includes the languages ROBOLAB and RIS.They are easy to learn and use programming languages. On the other hand, they are very limited in their access to the functionality of RCX and that is why, they are not suited for advanced programming.

2.3.2. Not Quite C

Not Quite C is denoted by NQC, moves ROBOLAB functionality to a C like language. In addition to this, it uses the same firmware on the RCX. It is possible to gain access to all functionality the firmware has to offer. So, more functionality can be obtained than with the graphical languages.

2.3.3. pbFORTH

pbORTH is such a programming language that, it is different from the majority of the other programming languages. Since it is interactive, user can also send a manual control commands to the RCX while the program is running. On the other hand, it requires a line-of-sight between RCX and the IR tower. Since pbFORTH is an interpreter running on board the RCX, it is not necessary to compile the movement commands before download them. Moreover, pbFORTH can be developed and run on any OS. The only thing is that, it requires a terminal emulator to send the commands to the RCX. To make this future useful in Matlab, Matlab should produce commands in FORTH syntax. However, pbFORTH uses a reverse Polish notation, which makes the code somewhat odd to look at and can give raise to some syntax errors.

2.3.4. legOS

In regards to functionality legOS is the ultimate language for the RCX. legOS gives the user full control over all the instructions in the processor and even single segments of the display. legOS works by replacing the existing firmware with its own Operating System. The user writes his program in C or C++ and then compiles it using a C/C++ cross compiler. The compiled program is then downloaded onto the RCX, which can contain 8 different programs.

2.3.5. TclRCX

TclRCX uses the language Tcl and can be run in Windows, Linux, Unix and MacOS. TclRCX is interactive, but unlike pbFORTH, TclRCX doesn't reverse polish notation. TclRCX uses the existing firmware.

2.3.6. Bot-Kit

Bot-Kit uses the Dolphin SmallTalk programming language. This language is an object oriented programming language, which needs Windows to run. Bot-Kit is also an interactive language, and it uses the existing firmware. Dolphin SmallTalk tool can be downloaded from the web. It is a user friendly graphical development tool. The language SmallTalk is quite suited for teaching object oriented programming and should be easy to learn and use.

2.3.7. BrainStorm

BrainStorm uses the language UCBLogo which is the Logo language adapted to LEGO MINDSTORMS. Logo is a dialect of Lisp. Logo developed specifically for controlling robots and runs in a special terminal window.

2.3.8. VB-like

This category covers languages such as BrickCommand, Gordon's Brick Programmer, MindControl, PRO-RCX and BotCode. The common features of these languages are, that they all use a Visual Basic like environment and the original firmware. They also require a Windows operating system.

Table 2.1. Lego Mindstorms programming environments

Tool	**Advantages**	**Disadvantages**
Graphical Languages (Robolab, RIS)	-Easy to learn and use	-It is difficult to make complex programs. -Very limited access to the functionality in the firmware
NQC	-C-like syntax -Access to all functions of the firmware	-Not real C-syntax -Uses original firmware
pbFORTH	-Interactive -No compiling necessary -Platform independent -Uses own firmware as interpreter	-Difficult to learn
legOS	-Complete processor control -ANSI C/C++ syntax -Uses its own firmware -Uses any regular C/C++ cross compiler	-Requires a cross compiler -Requires Linux
TclRCX	-Lots of OS's supported -Interactive	-Uses orginal firmware
Bot-Kit	-Very user friendly -Interactive	-Requires Windows -Uses original firmware
BrainStorm	-Developed specifically for controlling robots -Interactive	-Requires Windows -Uses original firmware
VB-like	-Easy if you know VB in advance	-Uses original firmware and Requires Windows

3. PATH PLANNING ALGORITHMS

Some of the most significant challenges confronting autonomous robotics lie in the area of automatic motion planning. The goal is to be able to specify a task in a high level language and have the robot automatically compile this specification into a set of low-level motion primitives, or feedback controllers, to accomplish the task. The prototypical task is to find a path for a robot, whether it is a robot arm, a mobile robot, or a magically free-flying piano, from one configuration to another while avoiding obstacles. From this early *piano mover's problem*, motion planning has evolved to address a huge number of variations on the problem., allowing applications in areas such as animation of digital characters, surgical planning, automatic verification of factory layouts, mapping of unexplored environments, navigation of changing environments, assembly sequencing, and drug deign. New applications bring new considerations that must be addressed in the design of motion planning algorithms [19].

Since actions in the physical world are subject to physical laws, uncertainty, and geometric constraints, the design and analysis of motion planning algorithms raises a unique combination of questions in mechanics, control theory, computational and differential geometry, and computer science. The impact of automatic motion planning, therefore, goes beyond its obvious utility in applications.

The classic path planning problem is the piano mover's problem [20]. Given a three dimensional rigid body, for example a polyhedron, and known set of obstacles, the problem is to find a collision-free path for the omnidirectional free-flying body from a start configuration to a goal configuration. The obstacles are assumed to be stationary and perfectly known, and execution of the planned path is exact. This is called *offline* planning, because planning is finished in advance of execution. Variations on this problem are the *sofa mover's problem*, where the body moves in a plane among planar obstacles, and the *generalized mover's problem*, where the robot may consist of a set of rigid bodies linked at joints, e.g., a robot arm.

The key problem is to make sure no point on the robot hits an obstacle, so we need a way to represent the location of all the points on the robot. This representation is the *configuration* of the robot, and the *configuration space* is the space of all configurations of the robot can achieve. An example of configuration is the set of joint angles for a robot arm. The configuration space is generally *non-Euclidian*, meaning that it does not look like an n-dimensional Euclidean space.

While dealing with a method to calculate the configuration space in which the vehicle can move, some questions should have an answer; the next big question is how exactly a path through it is planned? How does the robot get from an initial pose to a goal pose? The most common path planning methods are: Voronoi, Bug, Visibility Graph, Cell Decomposition, Potential Field, Probabilistic Roadmap and Rapidly Exploring Random Tree methods.

Complicated interactions between robots and the obstacles are caused by the arbitrary shapes of the real robots. It is necessary to simplify these complicated interactions.

In order to do this, a technique that is called the "Minkowski-Sum" is used. This method transforms the problem to one in which the robot can be considered as a point-object. The root idea is to artificially inflate the extent of obstacles to accommodate the worst-case pose of the robot in close proximity [21]. Figure 3.1 gives us the basic idea of the Minkowski-Sum methods.

If the robot can turn in all directions, then it can be thought as it has a circular shape for all the poses of the vehicle. The idea is to replace each object with a virtual object that is the union of all poses of the vehicle that touch the obstacle. It can be gotten the minimal Minkowski-Sum as the union of the obstacle and all the poses of the vehicle with vehicle nose just touching its boundary [21]. If the obstacle suitably extended and the vehicle is considered as a point object, it can be guaranteed that the vehicle will not hit the object.

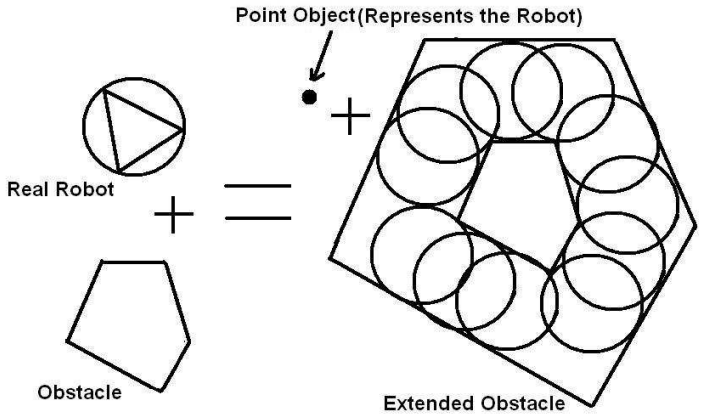


Figure 3.1. Minkowski Sum [21]

As it can be seen from the Figure 3.1, while inflating the obstacle, the Minkowski-Sum transforms the arbitrarily shaped vehicle to a point. The result is guaranteed free space outside the inflated object boundary.

3.1. Voronoi Methods

Voronoi diagrams are smart geometric structures that find applications throughout computer science. One is shown in Figure 3.2. The points on 2-D Voronoi diagram are equi-distant from the closest two objects on the workspace. In other words, Voronoi diagram of two points a, b is the line bisecting them and all the points on that line are equi-distant from a, b. On the other hand, the effective calculation of Voronoi diagrams can be a quite complex matter, but when given a set of polygonal objects, the algorithms can generate the appropriate equi-distant loci. Figure 3.2 also shows that, if the path given by Voronoi is followed, it is guaranteed to stay maximally far away from the closest objects [21].

Set of points on the diagram to find points that are closest to and visible from the start and the goal positions can be searched. Firstly, the robot is driven to the entry point on the Voronoi diagram and then it follows the path until it reaches the end point on the path where it leaves the path and drives directly towards the goal point.

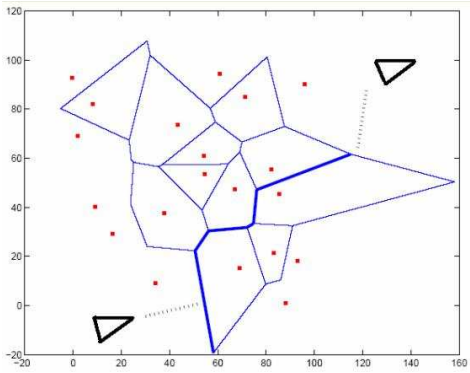
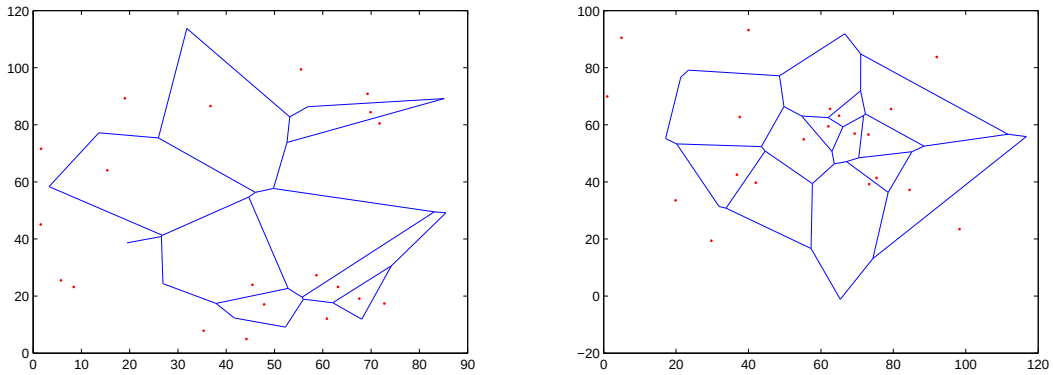


Figure 3.2. Concerning the point objects, a Voronoi diagram for obstacle avoidance [21]



(a) Voronoi Diagram - 1

(b) Voronoi Diagram - 2

Figure 3.3. Matlab implementation: Voronoi Method

Figure 3.3 shows two examples of the Matlab Simulations for the Voronoi Approach. These diagrams shows the paths that keeps the robot away from the point obstacles. If the robot follows one of these paths, it would stay far from the obstacles as much as possible while it tries to get the goal point.

3.2. Bug Methods

If Voronoi methods are used while proceeding to get from initial position to the final position upfront knowledge of the environment is required. In many cases this is unrealistic. Moreover, while the robot is driven to the Voronoi path, the vehicle is kept as far away from the obstacles as possible and this can have pros and cons. The robot may use the objects to plan its path, however; being away from the objects makes them

harder to sense. Secondly, the path which is generated using Voronoi can be extremely long and far from the shortest path.

Even a simple planner can present interesting and difficult issues. The Bug1 and Bug2 algorithms are among the earliest and simplest sensor-based planners with provable guarantees [22]. These algorithms assume the robot is a point operating in the plane with a contact sensor or a zero range sensor to detect obstacles. When the robot has a finite range sensor, then the Tangent Bug algorithm is a Bug derivative that can use that sensor information to find shorter paths to the goal [23]. The Bug and Bug-like algorithms are straightforward to implement; moreover, a simple analysis shows that their success is guaranteed, when possible. These algorithms require two behaviors; move on straight line and follow a boundary.

Perhaps, the most straightforward path planning approach is to move toward the goal, unless an obstacle is encountered, in which case, circumnavigate the obstacle until motion toward the goal is once again allowable. Essentially, the Bug1 algorithm formalizes the common sense idea of moving toward the goal and going around obstacles. The robot is assumed to be a point with perfect positioning with a contact sensor that can detect an obstacle boundary if the point robot touches it.

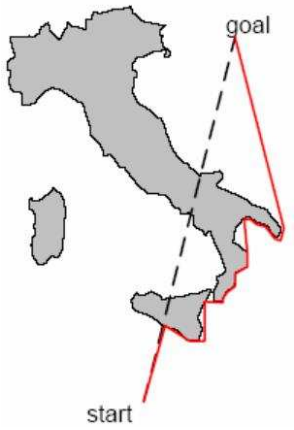


Figure 3.4. VisBug Algorithm [21]

This alternative family of approaches is called "Bug Algorithms" and it has some basic steps:

1. Draw a line **AB** from **A** to **B**. A refers to starting point and B refers to the goal pose. (AB may pass through obstacles that are known or as yet unknown)
2. Drive onto this line until either the goal is reached or an obstacle is hit.
3. If the robot hits an obstacle, circumnavigate its periphery until AB is met.
4. go to 2^{nd} step

Contrary to Voronoi approach, Bugs method keeps the vehicle as close to the obstacles as possible. On the other hand, the robot won't hit the obstacle because of the fact that, obstacle has been modified by the Minkowski-Sum. However, the path length can be extremely long. A more elegant modification would be to interchange step 3 in the original algorithm with "*on hitting and obstacle circumnavigate it's perimeter until* ***AB*** *is met or the line* ***AB*** *becomes visible in which case head for a point on* ***AB*** *closer to* ***B***" [21].

Figure 3.4 shows this kind of path which is generated by the modified algorithm is called "**VisBug algorithm**". Apparently, surrounding the object boundary is not always a good plan. On the other hand, interestingly it is guaranteed to get the robot to the goal location if it is indeed reachable.

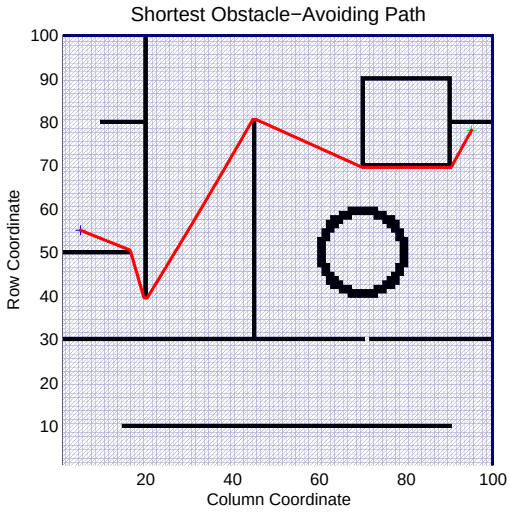


Figure 3.5. Matlab implementation I: VisBug Algorithm

Figure 3.5 and Figure 3.6 show the Matlab simulations of the VisBug Algorithm. The robot's starting point is marked by green plus while the goal point is marked by

blue point. As it can be seen from these simulations, robot reaches to the goal point successfully without hitting any of the obstacles.

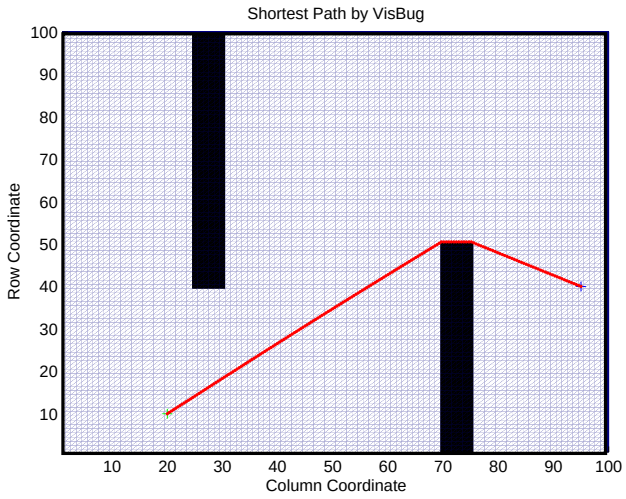


Figure 3.6. Matlab implementation II: VisBug Algorithm

3.3. Potential Field Method

One of the path planning methods is the so called "Potential Field Method". Potential Field Method is a very popular path planning method and the idea in this method is again very simple. The goal is treated as a point of low potential energy while the obstacles as areas of high potential energy. In addition to this, the robot is assumed as a ball-bearing free to roll around on a "potential terrain" then it will naturally pass around the obstacles and down towards the goal point.

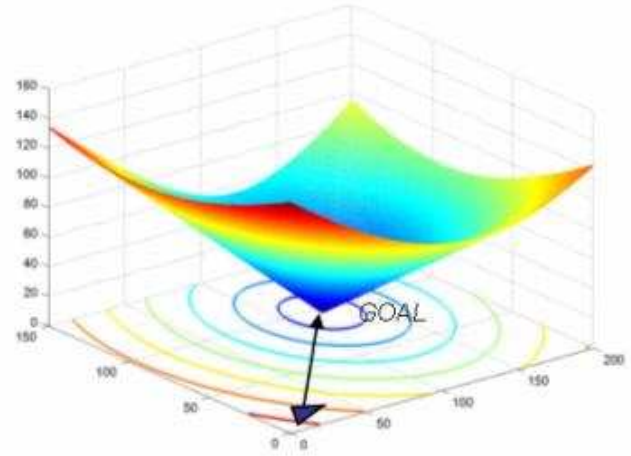
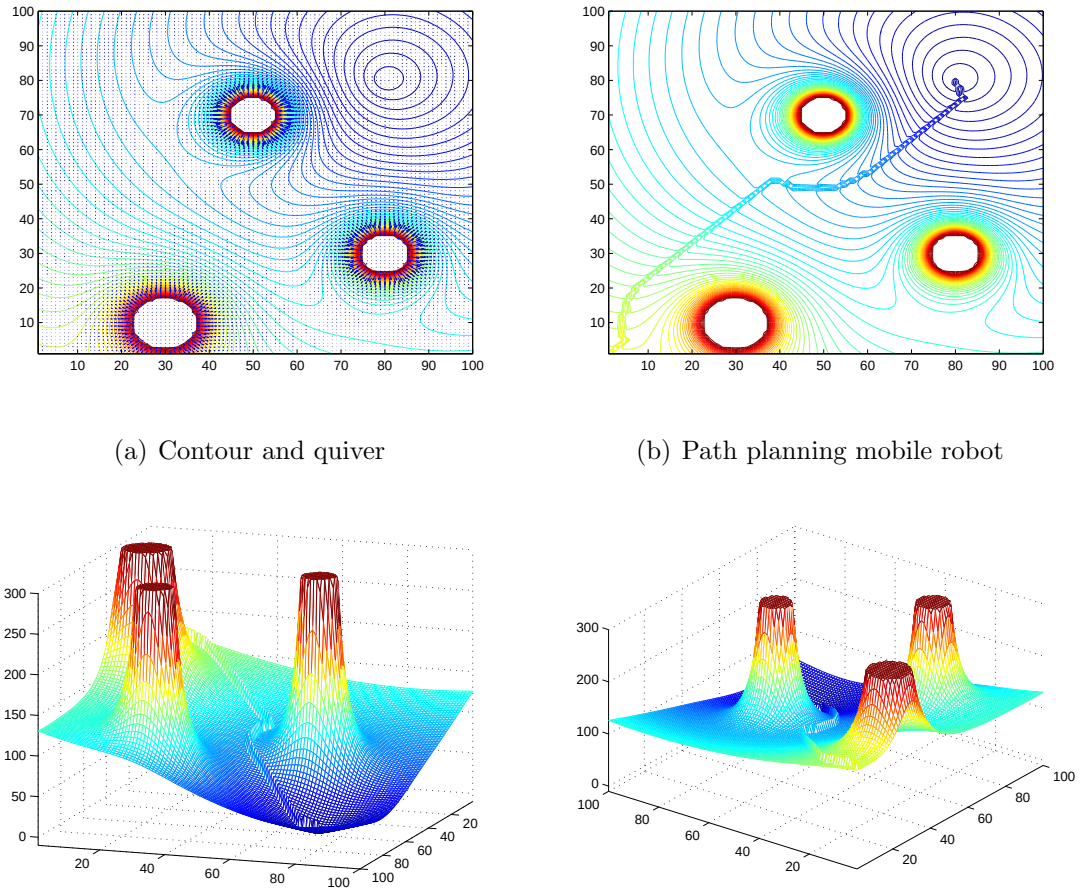


Figure 3.7. Potential Field Approach [21]

It is not clear, what exactly the potential function look like. There is no single answer to this question. Making the potential of an obstacle is an inverse square function of the distance between vehicle and the obstacle would be a good choice. So that wherever the robot starts, in the absence of obstacles, it will fall towards the goal point; it may be chosen everywhere-convex potential for the goal.



(a) Contour and quiver

(b) Path planning mobile robot

(c) Path planning mobile robot

(d) Path planning mobile robot - reverse angle

Figure 3.8. Matlab implementation I: Potential Field Method

Figure 3.8 and Figure 3.9 show the Matlab simulations of the Potential Field Method. In Figure 3.8 the first graph shows the contour and quiver while the other three graphs show the calculated trajectory from three different views.

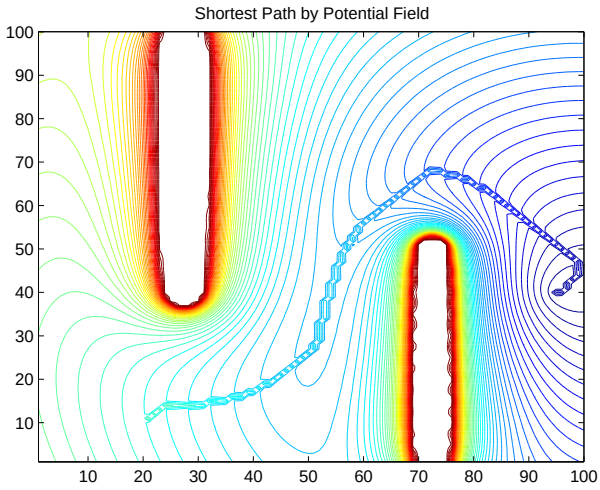


Figure 3.9. Matlab implementation II: Potential Field Method

To sum up;

- Robot is treated as a point under the influence of an artificial potential field.
- Generated robot movement is similar to a ball rolling down the hill
- Goal generates attractive force
- Obstacles are repulsive forces

Although, potential field method is very useful for the path planning, it has one serious drawback that is, it only acts locally. The vehicle simply reacts to local obstacles and always moves in a direction of decreasing potential. Sometimes, it can cause the vehicle getting trapped in a local minimum. As it can be seen from Figure 3.10, the vehicle will descend into local minima in front of the two features and will stop, because of the fact that any other motion will increase its potential. When the code is run, it can be seen that the vehicle gets stuck between two features from time to time. In order to overcome this drawback, after detecting the trap, the goal point can be temporarily moved to out of the trap.

The potential method is someway between the Bug method and Voronoi method in terms of distance to obstacles. The Bug algorithm sticks close to them, the Voronoi as far away as possible. The potential method simply directs the vehicle in a straight line to a goal unless it moves into the vicinity of an obstacle in which case it is deflected.

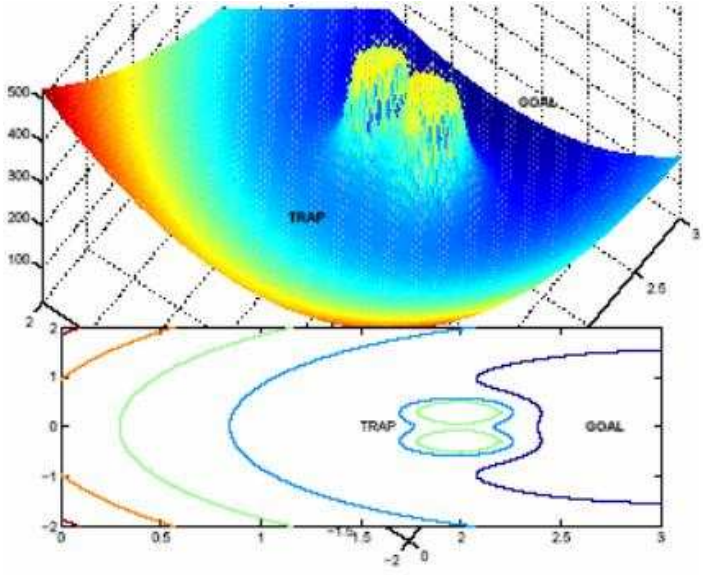


Figure 3.10. The vehicle can get stuck in a local-minima or "trap" [21]

3.4. Visibility Graph Method

The defining characteristic of a visibility map are that its nodes share an edge if they are within line of sight of each other, and that all points in the robot's free space are within line of sight at least one node on the visibility map. This second statement implies that visibility maps, by definition, possess the properties of accessibility and departability. Connectivity must then be explicitly proved for each map for the structure to be a roadmap [19].

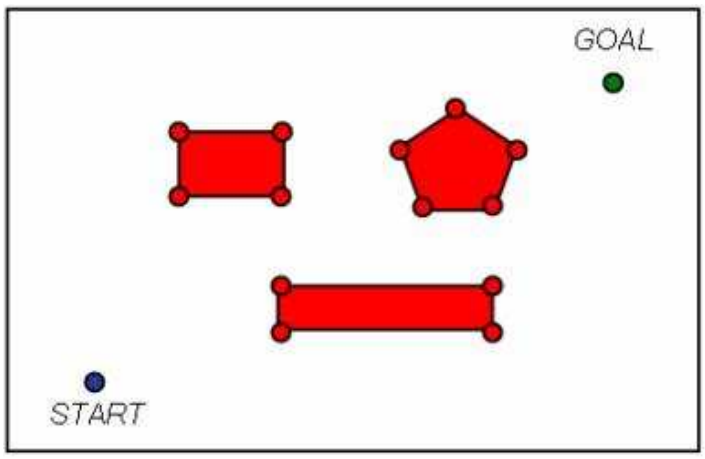


Figure 3.11. Model obstacles as polygons [24]

In order to model robot as a point, again The Minkowski Sum is used. In other words, the standard practice is to grow the obstacles by convolving them with the

robot's dimensions. The standard visibility graph is defined in a two-dimensional configuration space.

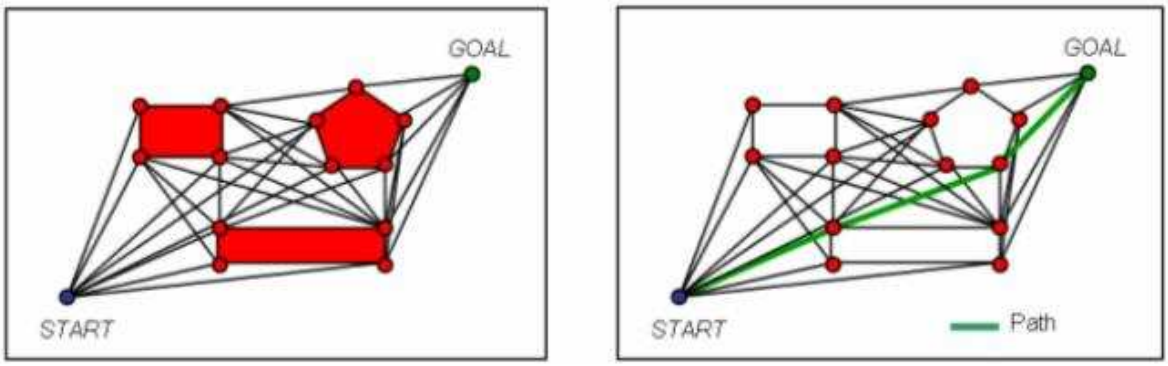


Figure 3.12. The shortest path using Dijkstra's Algorithm [24]

Then a graph G(V,E) is constructed. All polygon vertices are added to V, as are the start and goal positions. The each vertices are connected so that they are visible to one another with lines. And, these lines are added to E. The polygon edges are also added to E. Then, the only thing to do is to find the shortest path from the initial position to the goal pose in G. As it can be seen from Figure 3.11 and Figure 3.12, the shortest path can be found using Dijkstra's Algorithm.

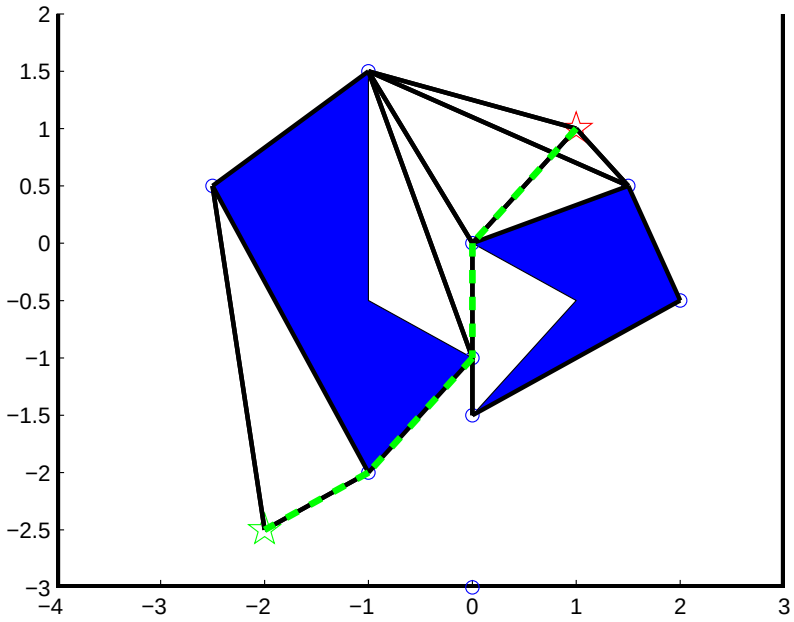


Figure 3.13. Matlab implementation I: Visibility Graph (Dijkstra's Algorithm)

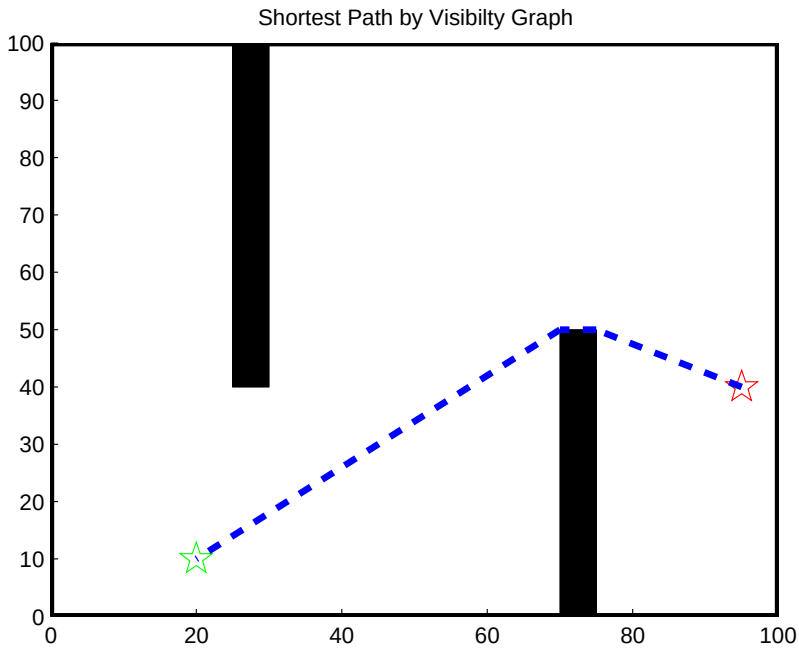


Figure 3.14. Matlab implementation II: Visibility Graph (Dijkstra's Algorithm)

Figure 3.13 and Figure 3.14 show the Matlab simulations of the Visibility Graph (Dijkstra's Algorithm). The starting point is shown by green star while the goal point is shown by red star. In Figure 3.13, the thin solid lines delineate the edges of the visibility graph for the two obstacles represented as filled polygons. The thick dotted line represents the shortest path between the start and goal.

3.5. The Cell Decomposition Approach

The principal idea behind this method is that a path between the initial coordinate and the goal pose can be determined by subdividing the free space of the robot's configuration into small regions called cells. After the decomposition, a connectivity graph is obtained according to the adjacency relationships between the cells, where the nodes represent the cells in the free space, and links between the nodes show that the corresponding cells are adjacent to each other. It is possible to determine a continuous path from this connectivity graph, by simply following adjacent free cells from initial point to the goal point. Figure 3.15 clearly explains the procedures using both exact cell decomposition and approximate cell decomposition method.

3.5.1. Exact Cell Decomposition

We consider a different type of representation of the free space called an *exact cell decomposition*. These structures represent the free space by the union of simple regions called *cells*. The shared boundaries of cells often have a physical meaning such as a change in the closest obstacle or a change in the line of sight to surrounding obstacles. Two cells are adjacent if they share a common boundary. An adjacency graph, as its name suggests, encodes the adjacency relationships of the cells, where a node corresponds to a cell an edge connects nodes of adjacent cells.

Assuming the decomposition is computed, path planning with a cell decomposition is usually done in two steps: first, the planner determines the cells that contain the start and goal, respectively, and then the planner searches for a path within the adjacency graph could serve as a roadmap of the free space as well. Therefore, mapping can be achieved by incrementally constructing the adjacency graph.

Cell decompositions, however, distinguish themselves from other methods in that they can be used to achieve coverage. A coverage path planner determines a path that passes an effector (e.g., a robot, a detector, etc.) over all points in a free space. Since each cell has a simple structure, each cell can be covered with simple motions such as back-and-forth farming maneuvers; once the robot visits each cell, coverage is achieved. In other words, coverage can be reduced to finding an exhaustive walk through the adjacency graph. Sensor based coverage is achieved by simultaneously covering an unknown space and constructing its adjacency graph [19].

In this method, the first step is to decompose the free space, which is bounded both externally and internally by polygons, into trapezoidal and triangular cells by simply drawing parallel line segments from each vertex or each interior polygon in the configuration space to the exterior boundary. Then a number is assigned to each cell and each cell is represented as a node in the connectivity graph. The path is obtained in this graph corresponds to a channel in free space, which is illustrated by the sequence of striped cells. Then this channel is translated into a free path by connecting the

initial coordinates to the goal pose through the midpoints of the intersections of the adjacent cells in the channel.

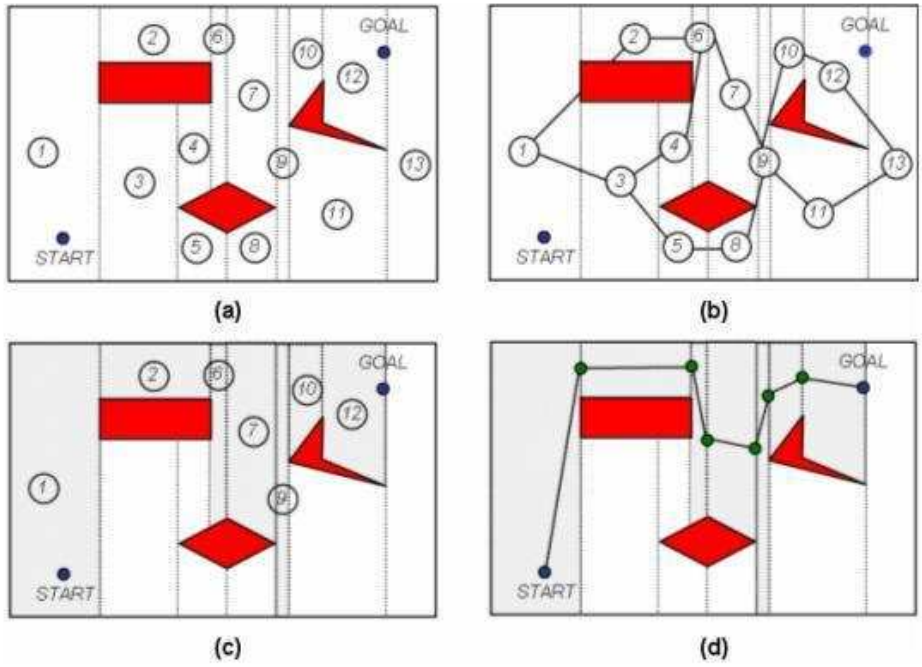


Figure 3.15. Cell Decomposition Approach [24]

3.5.2. Approximate Cell Decomposition

This approach is different from the exact cell decomposition method because it uses a recursive method to continue subdividing the cells until one of the scenarios that are listed below occurs:

- Each cell lies either completely in free space or completely in the C-obstacle region.
- An arbitrary limit resolution is reached

It stops decomposing, if a cell fulfills one of these criteria. "Quadtree" decomposition is the other name of this method because a cell is divided into four smaller cells of the same shape each time it gets decomposed. After the decomposition steps, the free path could be found easily by following the adjacent, decomposed cells through free space as it can be seen from the Figure 3.15.

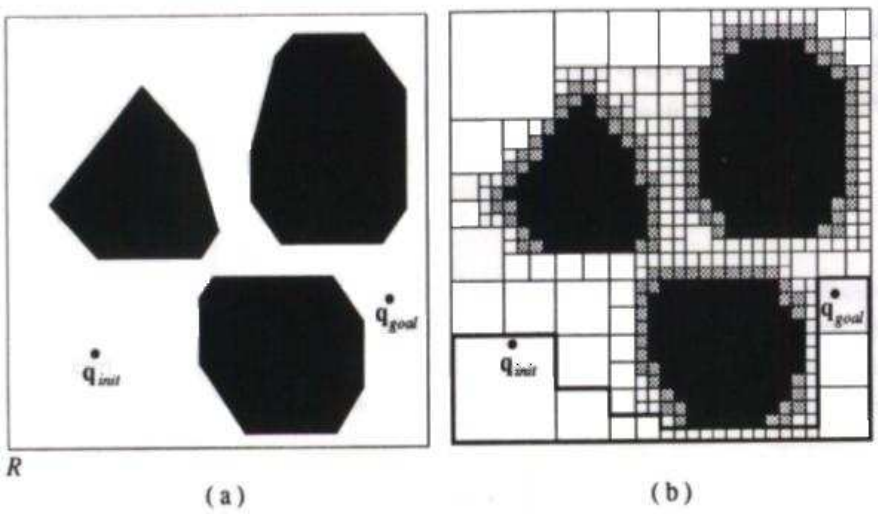


Figure 3.16. Approximate Cell Decomposition [24]

Both types of cell decomposing methods have advantages as well as disadvantages. In the former method, if a free path exists, exact cell decomposition will find it; however, the computation for this accuracy requires more difficult mathematical process. Approximate cell decomposition is less involved, but can yield similar, if not exactly the same, results as exact cell decomposition.

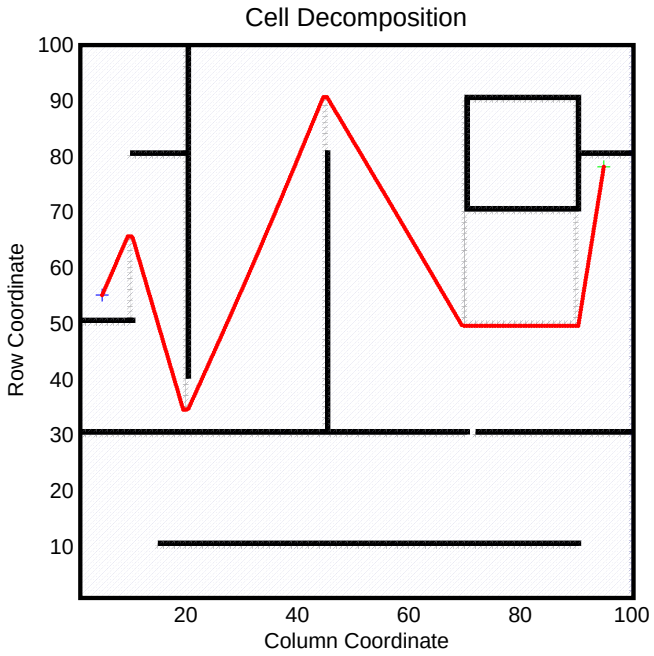


Figure 3.17. Matlab implementation I: Cell Decomposition

Figure 3.17 and Figure 3.18 show the Matlab simulations of the Cell Decomposition Method. The starting point is shown by green "+" while the goal point is

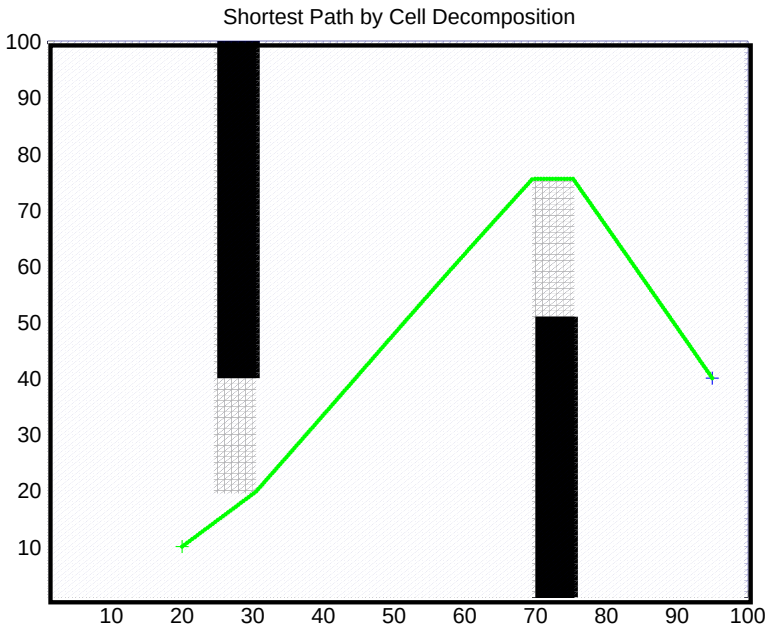


Figure 3.18. Matlab implementation II: Cell Decomposition

shown by blue "+". The thick red line represents the path which is calculated by Cell Decomposition Approach between the start and goal.

3.6. Probabilistic Roadmap

The probabilistic roadmap (PRM) has been one of the most popular path planning methods during the past decade, especially when the problem is characterized with many degrees of freedom. On the hand, in situations where the robot has to pass through narrow ways and tight spaces between obstacles, the implementation of this method encounters difficulties [25].

A classic PRM planner samples at random the robot configuration space to construct a network, called a roadmap that captures the connectivity of the free space. PRM planners are not only simple to implement, but also very efficient to use. Therefore, PRM planners have been adapted into many applications, including robotics, virtual prototyping, computer animation and computational biology. However, the PRM method causes some undesirable situations where the robot has to pass narrow passage in the workspace. This is caused by the uniformity of the sampling used in

the planner; it places many samples in large open regions and too few in tight passages. This disadvantage does not occur in cell decomposition methods, but the cell decomposition methods are only applicable in low dimensional configuration spaces [26].

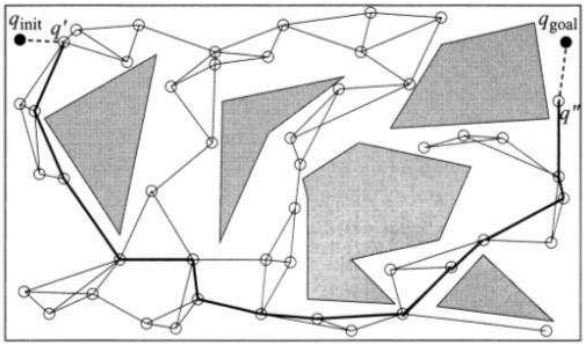


Figure 3.19. Example of Probabilistic Roadmap [27]

Figure 3.19 shows an example of how to solve a query with the roadmap. The configurations qinit and qgoal are first connected to the roadmap through q' and q". Then a graph-search algorithm returns the shortest path denoted by the thick black lines.

This method proceeds in two phases: a learning phase and query phase. In the learning phase, a probabilistic roadmap is constructed and stored as a graph whose nodes correspond to collision-free configurations and whose edges correspond to feasible paths between these configurations. These paths are computed using simple and local planner. In the query phase, any given start and goal configurations of the robot are connected to two nodes of the roadmap; the roadmap is then searched for a path joining these two nodes. This method can be applied to virtually any type of holonomic robot since it is general and easy to implement.

Current sampling-based planners can be classified into two sets as follows. The first, referred to as multiple-query planners, are characterized by the ability to answer multiple queries for the same environment quickly at the account of substantial preprocessing time consumption. The PRM can be considered as an example of such a planner. The second, referred to as single-query planners, are based on importance sampling. The planners in this set concentrate samples in a nonuniform way. For in-

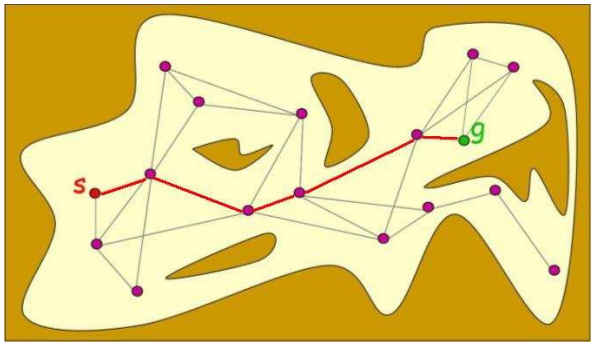


Figure 3.20. Probabilistic Roadmap [27]

stance, along the configuration space boundaries or the medial axis, and are primarily designed for solving problems with narrow passages. Particularly, techniques based on the Rapidly-exploring Random Trees (RRTs) have generated highly successful single-query planners. Despite RRTs work well on many problems, they have weaknesses, which cause them to explore slowly when the sampling domain is not well adapted o the problem [28].

To a large extent, the Visibility PRM reduces the number of nodes in the roadmap. This property allows better performance due to the smaller search space when the Visibility PRM building the edges. On the other hand, Visibility PRM only adds nodes when they are not straight line visible to any other nodes or connect two previously separate connected components.

It is noteworthy that times for RRT algorithms are for their plan stage while times for PRM algorithms are for their construction stage. The plan phase for the PRM algorithms is always less than the construction time, which can be quicker than RRT algorithms.

3.7. Rapidly-Exploring Random Tree

A Rapidly-exploring Random Tree (RRT) is a data structure and can be described as an algorithm that is designed for efficiently searching nonconvex high-dimensional spaces. RRTs are constructed by a progressive stage and this stage quickly reduces the expected distance of a randomly chosen point to the tree. RRTs can be used as a solu-

tion for the path planning problems that involve obstacles and differential constraints. RRTs can also be described as a technique for generating open-loop trajectories for nonlinear systems with state constraints. Generally, an RRT alone is not sufficient to solve a planning problem. Therefore, RRT can be defined as a component that can be incorporated into the development of a variety of different planning algorithms [2].

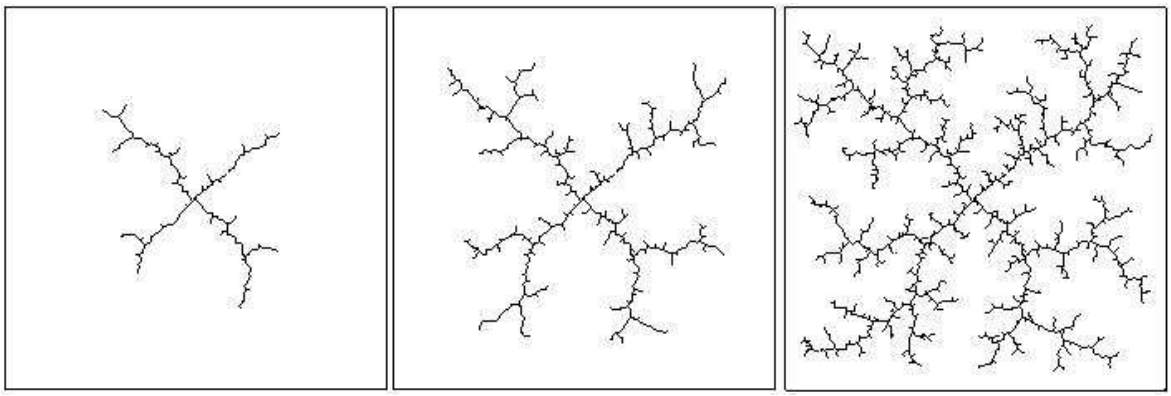


Figure 3.21. Rapidly Exploring Random Tree [29]

As it can be seen from the Figure 3.21, the RRT quickly expands in a few directions to quickly explore the four corners of the square. On the other hand, it is not an easy task to find a method that yields such desirable behavior even if the construction method is simple. For instance, absolute random tree is constructed incrementally by selecting a vertex at random, an input at random, and finally applying the input to generate a new vertex. Even though one might intuitively expect the tree to "randomly" explore the space, a very strong bias toward places already explored exists. As it can be seen from the simulations, a random walk also suffers from a bias toward places already visited. On the other hand, an RRT works in the opposite manner by being biased toward places not visited before. This can be observed by considering Voronoi diagram of the RRT vertices since larger Voronoi regions occur on the frontier of the tree. Because vertex selection is biased on the nearest neighbors, this causes that vertices with large Voronoi regions are more likely to be selected for expansion. Basically, an RRT is created by iteratively breaking large Voronoi regions into smaller ones [28].

RRTs are grown from initial and goal points for the following cluttered 2D environment.

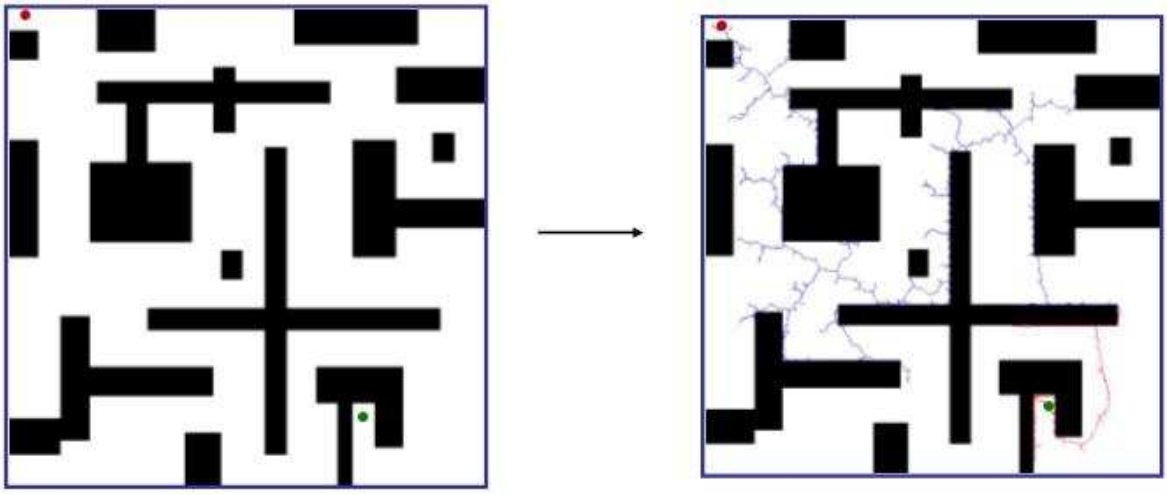


Figure 3.22. RRT construction [29]

In Figure 3.22, blue path represents the RRT that is grown from the initial point while red path represents the RRT that is grown from the goal point. Finally, the two RRTs meet in the middle, and the following path is obtained.

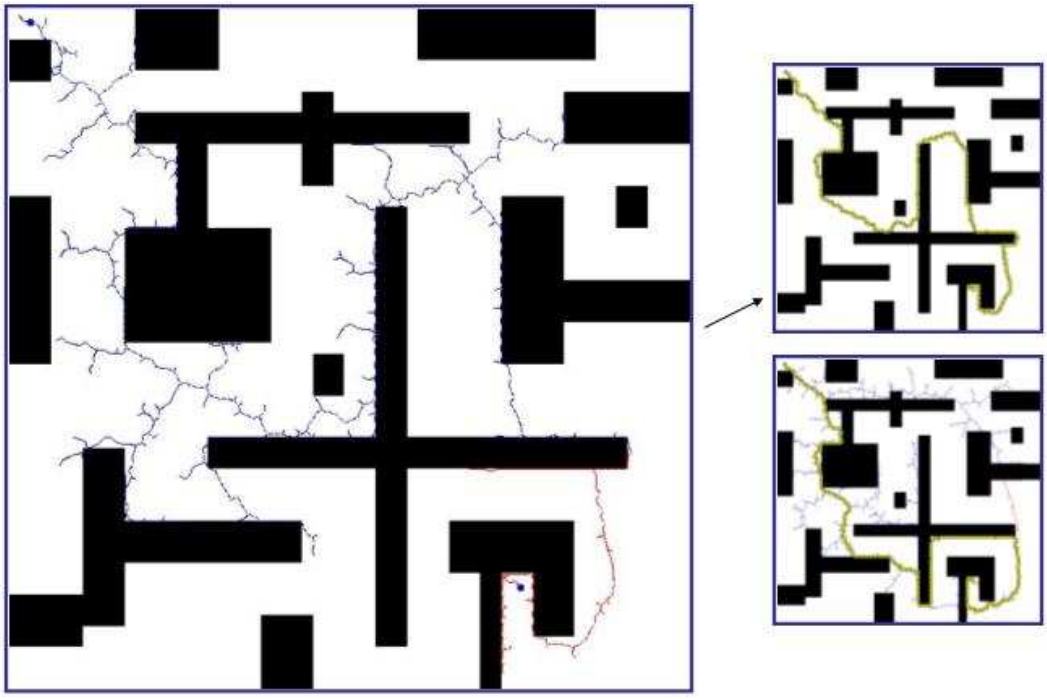


Figure 3.23. RRT map forming [29]

As it can be seen from Figure 3.23, a different path is obtained in a separate run because of the randomness.While determining how to get from one place to another place in a 2D cluttered environment, RRTs can be used to have a solution. In order to achieve this problem, there are also some other techniques.

3.8. Comparison of Path Planning Algorithms

Path planning algorithms play an important role in motion strategy because they form the main body of the whole program that determines how the robot should move from it's initial position to the goal position without hitting any obstacle and make sure the path is short. The five algorithms that is to be compared are:

1. Voronoi Approach
2. Bug Approach
3. Visibility Graph Approach
4. The Cell Decomposition Approach
5. The Potential Field Approach

All of these algorithms need a pre-determined map of obstacles, an initial point and a goal point. They do not have the ability to look ahead for unidentified obstacles. Although these algorithms are efficient in finding a path, there are some advantages and disadvantages provided by one algorithm over the others. Table 3.1 compares five most common path planning methods.

Table 3.1. Comparison of path planning methods

	Potential Field	**Approx. Cell**	**Voronoi**	**Visibility Graph**	**VisBug**
Practical above 2D ?	Good	Good			
Fast to Compute ?	Yes	Yes	Yes	In 2D	Yes
Usable Online ?	Yes				Yes
Gives Optimal ?			Yes	In 2D	Yes
Easy to Implement ?					Yes

In addition to this, The Bug Approach, The Visibility Graph, The Cell Decomposition Approach and The Potential Field Method's Matlab simulation results were compared and shown in one graph that can be seen in Figure 3.24. Furthermore, Table

3.2 shows the comparison of path planning methods in terms of path lengths that are measured from the Figure 3.24.

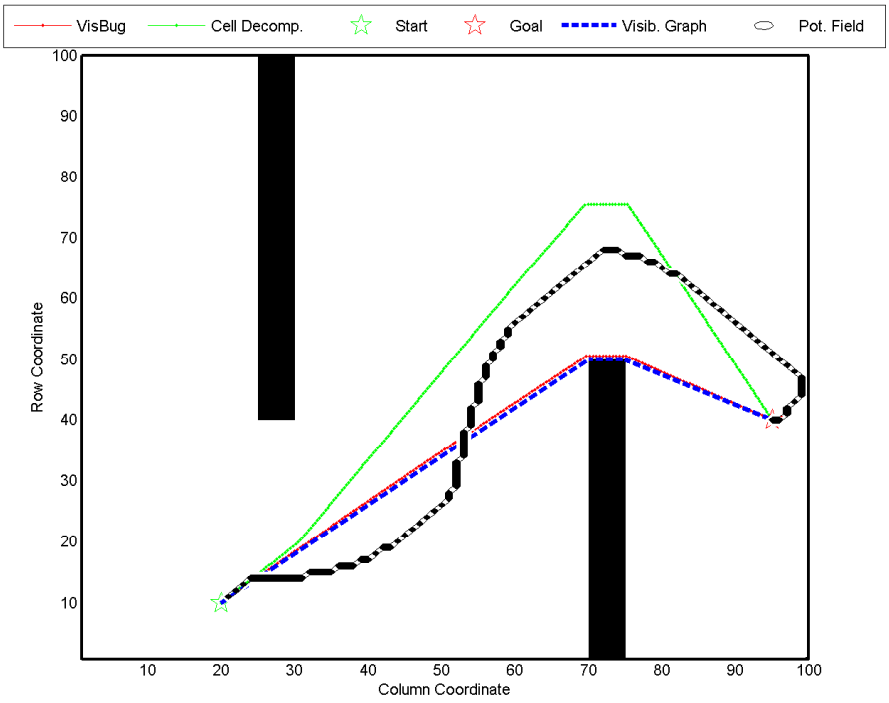


Figure 3.24. Comparison of path planning methods

In this project, our challenge was to be able to solve the path planning problem fast enough to drive the robot in our workspace. The VisBug algorithm indeed achieved this goal in our experiments where the environment is composed of several fixed obstacles and a mobile Lego robot with two motors moving independently. According to the Table 3.2, VisBug Algorithm gives us the shortest path length among the other simulated algorithms.

Table 3.2. Comparison of path lengths

Path Planning Methods	**Path Lengths**
Cell Decomposition	11.15 cm
VisBug Algorithm	8.55 cm
Visiblity Graph	8.55 cm
Potential Field	11.75 cm

4. DEVELOPMENT STAGE OF THE PROGRAM

The Lego robot consists of RCX microcomputer, two motors and different types and sizes of Lego pieces that are included in Lego Mindstorms Robotics Invention System 2.0.

4.1. Programming Steps

After examining the most common path planning approaches' simulations in Matlab, we have decided to use VisBug Algorithm as the Shortest Path Algorithm into the program written in Microsoft Visual C++. Because, VisBug Algorithm seems to be the most appropriate technique for our study. It is easy to be programmed and it finds the shortest path contrary to Potential Field and Cell Decomposition Methods. In addition to this, Voronoi Method doesn't fit to our study, since we are not dealing with the point while obstacles. Our obstacles are 2D objects.

Upon initiation the empty field picture is being sampled and in this phase obstacles are being recognized. This is done only once. Then, 5 stages program begins to run. The five stages are as follows;

1. Picture sampling
2. Objects separation
3. Recognition of the robot and the target
4. Path planning
5. Robot's movement control

4.1.1. Main Program

The main program is written in Microsoft Visual C++ (MFC) and it includes two ActiveX modules. These ActiveX controls and their roles are explained in the following sections.

4.1.2. VideoOCX ActiveX Control

VideoOCX is an ActiveX control that allows programmers to easily integrate video capture and image processing capabilities into their software applications. This module, which can be downloaded from **http://www.videoocx.de/**, belongs to Marvelsoft Company and it supports video capture and manipulation.

4.1.3. Phantom ActiveX Control

This ActiveX control, which can be downloaded from **http://members.cox.net/pbrick-alpha/Phantom.htm**, supports the RCX IR communication. In addition to this, it makes using USB IR tower possible.

4.1.4. Robotic Command Explorer

RCX programmable brick is the heart of the MindStorms kit. In order to take input from the environment and to process the data, the RCX uses sensors and it signal up to three motors to turn on and off in forward or reverse at any of 8 different speed level. This brick can also process over 1000 commands per second. It uses an 8-bit Hitachi H8/3297 microcontroller running at 16 MHz. It has 32K of external SRAM to download programs, 16K of ROM and 512 bytes of SRAM for downloading the firmware.

The RCX brick also has an LCD display and an IR-transceiver that is used to download programs and communicate with other bricks. The RCX unit is built for the easy attachment of Lego building blocks and pieces.

The RCX programmable brick comes with original firmware. In order to run the main program, **MS_THESIS.NQC**, written in BrickCC, should be downloaded to the RCX with the correct firmware "**firm0328.lgo**" which can be downloaded from **http://mindstorms.lego.com/**. After that, it would be possible to make some serial communication between the RCX and the computer.

Figure 4.1. RCX [30]

4.1.5. Buttons on User Interface and Their Roles

The source codes which were written using C++ are working under the user interface which can be seen in Figure 4.2. There are 7 buttons on the user interface. Three of them are related to the robot's movement control and the others are related to the some video and software settings that are necessary to obtain and capture the working environment.

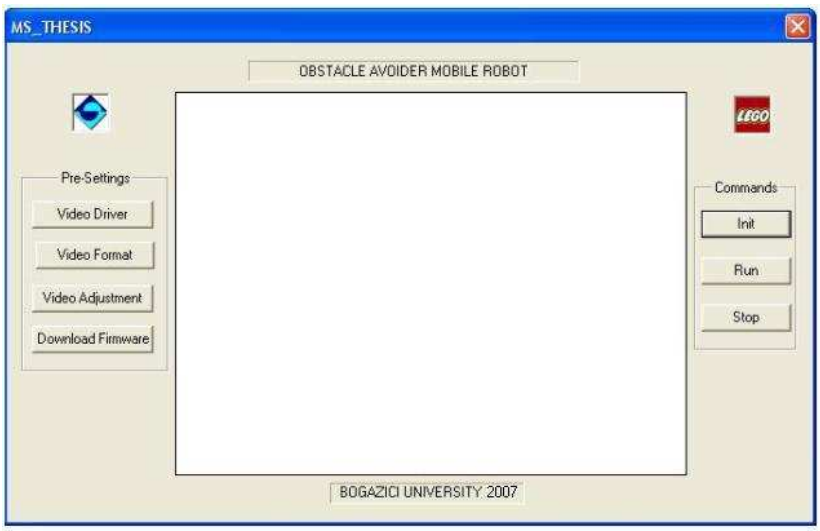


Figure 4.2. The user interface

- Video Driver: Video driver selection is made via this button.
- Video Format: Resolution, pixel depth settings.
- Video Adjustment: Brightness, contrast, color settings.
- Download Firmware: Downloads the firmware to the RCX.
- Init: Starts the application. Before pressing this button,it must be sure about that the environment should only have the obstacles and the borders not the robot and the target in this stage. The robot and the target will be placed later.
- Run: Runs the program and makes the robot move.
- Stop: Stops the application and robot's movement.

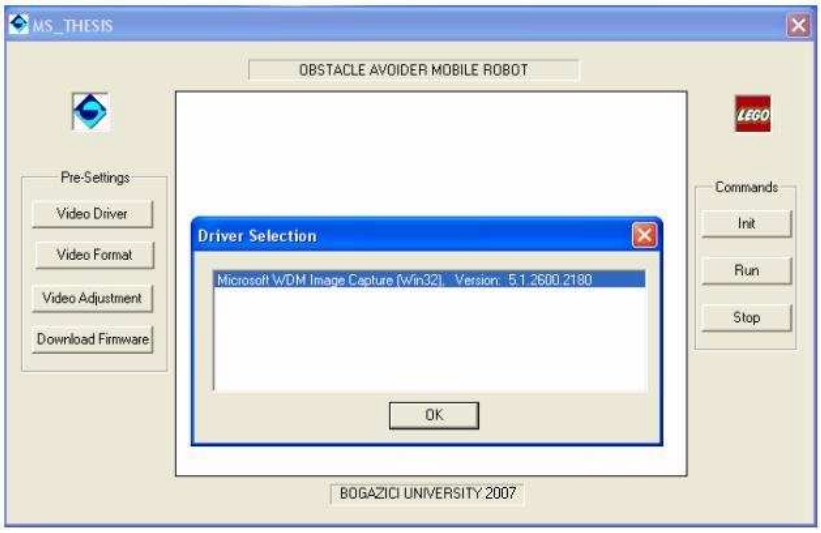


Figure 4.3. Video driver selection

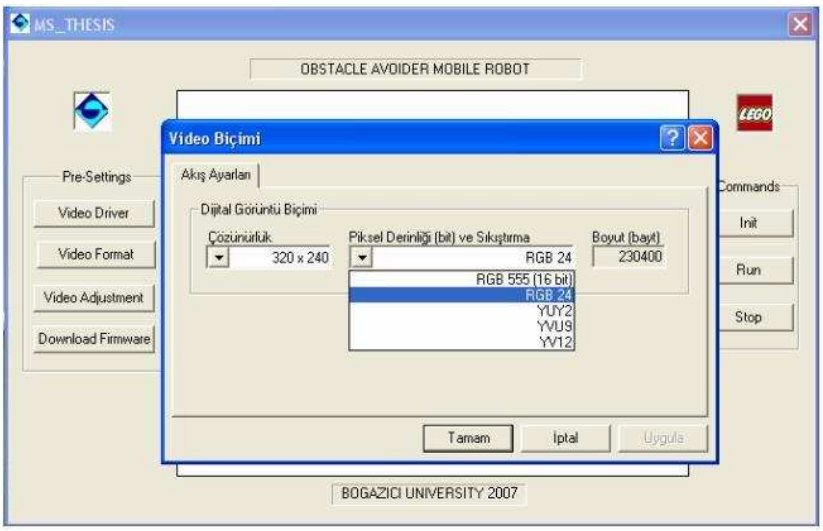


Figure 4.4. Video format settings

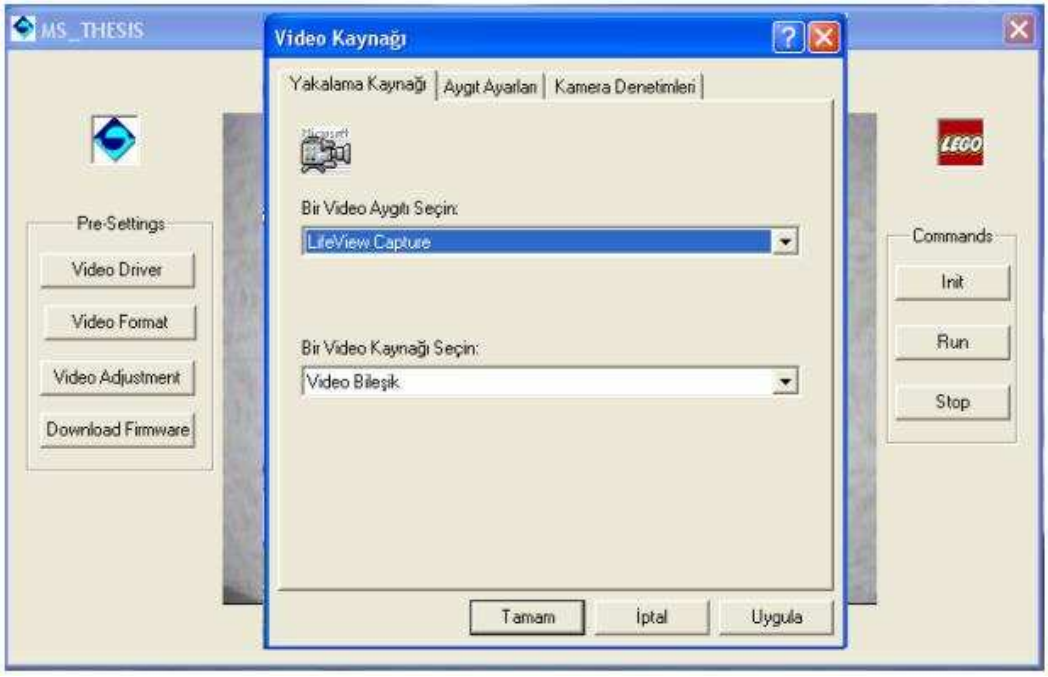


Figure 4.5. Video source selection

4.2. Functions in Main Program and Their Roles

The following programs and functions are embedded in the main program and they are mainly controlling the MFC application.

- MS_THESISDlg - The program controls the dialog.
- Rover - The program is responsible for the robot control.
- List - The program includes the properties of the objects.
- CVideoOCX - The program that makes VideoOCX ActiveX control working.
- CPhantomCtrl - The program that makes Phantom ActiveX control working.

Important functions in CGameDlg :

- OnInit - Determines the background and the obstacles.
- OnRun - Initiates movement.
- FindObj - Finds obstacles and keeps them into memory.
- GetStat - Determines the color of the objects.
- RGBtoHSV - Shows pixels and converts RGB color to HSV color space.

- ExecuteMoves - Controls the dialog and sends the controls commands to the robot.
- DrawBull - Determines the vehicle and the target's centers of mass, their red points and puts blue "+" to the the next move.

Important functions in Rover :

- GetAngle - Calculates the angle between two points.
- GetData - Holds the list of the objects and compares it to determine the robot and the target's coordinates. It also calculates the angle that goes to the target via GetAngle. It recognizes the catching moment and in case of an unexpected situation it pauses the dialog.
- IsPathClear - It checks whether or not there is an obstacle between target and the robot.
- Where2go - If there is an obstacle on the route, it determines the optimal next move.
- Chase - Controls the robot's movement.

Important functions in Pixel_List :

- Pixel_Insert - Downloads the pixel values.
- GetFirst - Carries pixels from one list to another.
- GetPixelFrom - Transfers pixels one list to another.
- GetCoord - Sends statistical values to the header.

Important functions in Object_List :

- Object_Insert - Downloads new pixels to the Object List.
- Object_Delete - Deletes objects from the list.

4.3. Workspace

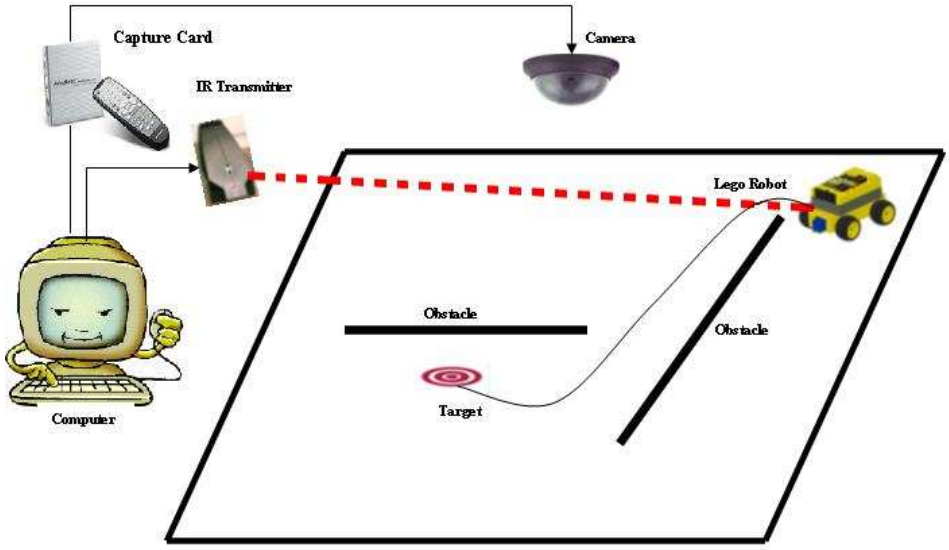


Figure 4.6. Workspace

Requirements for this study are;

- Lego Mindstorms: Robotics Invention System 2.0 to build the robot and the target. USB IR Transmitter to send the control commands
- Dome type camera to acquire the vision
- AVerMedia AVerTV USB2.0 Lite External TV Card for video capture
- Windows Laptop
- Software: Matlab 7.0 for the path planning algorithm simulations and Microsoft Visual C++ 6.0 for the main program and the user interface.The User Interface includes 2 ActiveX modules:
  - VideoOCX : Supports video capture and manipulation
  - Phantom : Supports the RCX IR communication

4.4. Object List

Using a data list is always advantageous because of the fact that it includes some data about the objects which are located in the working environment. It includes objects nodes for each of the objects and each of the object nodes keeps the pixel list and pixel node.

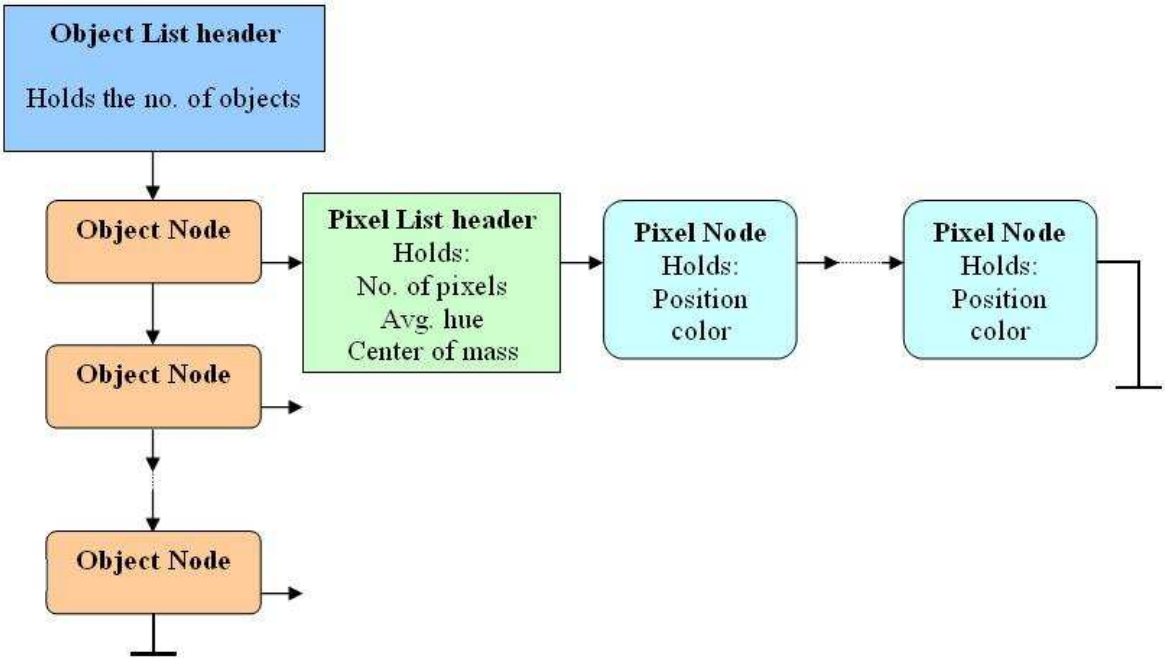


Figure 4.7. Basis of the pixel list and the object list

In addition to this, each pixel list holds the number of pixels, average hue and the center of the mass while the pixel node holds the position and the color of an object.

We subtract the background picture from the sampled picture. The difference matrix is being scanned pixel by pixel. If one of the pixel's components (R,G or B) exceeds a certain threshold we save it in a list and then we check its neighbors using the "Region Growing" method. When a pixel enters the list we transform its absolute RGB values to one Hue value by means of RGB to HSV transformation. We eliminate small lists and save each object's attributes: size, hue and position of the center of mass.

5. THE USER INTERFACE

MS-THESIS.EXE is the name of user interface written in Microsoft Visual C++. It is also called a Microsoft Foundation Class application, which is denoted by MFC and the user interface includes two ActiveX controls. C++ serves a suitable platform for Real Time applications. All the controls were made by Debugger during the programming.

Figure 5.1. Empty field sample

After initiation, the view of empty field is sampled. At this stage obstacles are determined. This is nonrecurring stage. It is done only once.

In the initiation stage, the picture of empty space is obtained so that the difference image, which is between the initiation picture and the picture that are obtained during the execution of the program, can be calculated. Black - White copies are obtained from this empty picture. Moreover, obstacles are included into the initiation picture. If the program determines an obstacle between the robot and the target, then it should determine an optimum route in order to let the robot reach the target. It has been decided to work with low resolution pictures because of the program speed is not so high.

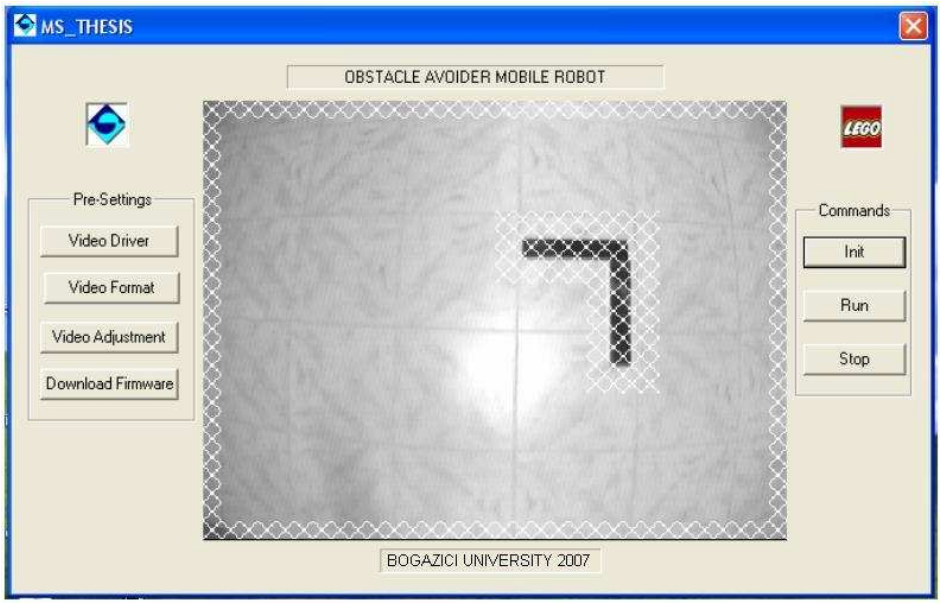


Figure 5.2. Obstacles and borders are determined and marked with white X

In the program, "1" represents the place of obstacles and "0" represents the free space where the robot can move freely into this space. Furthermore, borders of the picture frame are also marked as obstacles. Every pixel into the picture is 10x10 in size. It is possible to process a picture with this size in an optimum time period.

The schema below summarizes the algorithm which is used by the program:

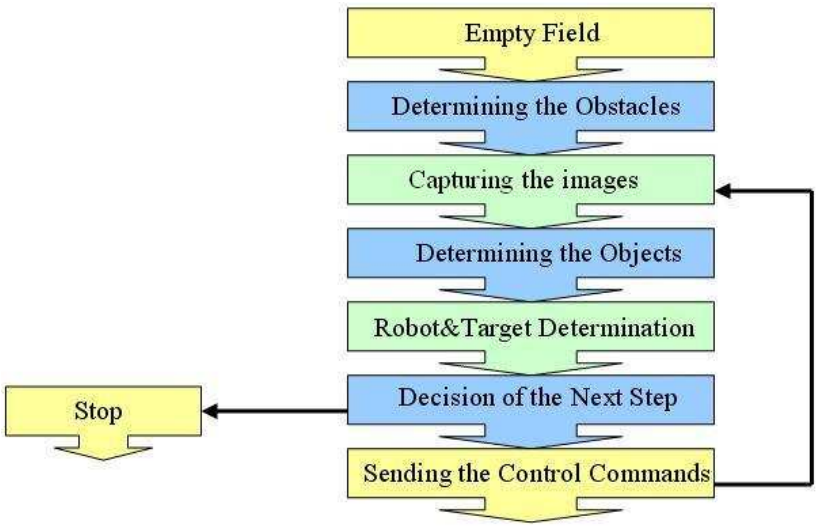


Figure 5.3. Motion algorithm

5.1. Picture Sampling

Some serious drawbacks occur while determining the objects. If good quality pictures are to be obtained during the image processing, then acceptable algorithm running speed must be obtained. Although the better picture quality provides the easier object recognition, it slows the algorithm speed down. This constraint has to be calculated precisely so that the reasonable result can be obtained.

There are some features that affect the quality of the pictures;

- The number of pixel,
- Colored or Black & White picture,
- Color density in each pixel.

- Selected parameters : 320x240 - In order to determine the size of the objects, this resolution value was selected. As long as the number of pixel is not so high, pixel number of the objects should be high enough so that noise in the images can be eliminated. According to this resolution (320x240), the pixel number of the objects is considered between 200-500 and each picture is scaled to have 76800 pixels.
- Color : RGB images can be obtained by means of a dome camera and this format can recognize the objects easier. This colored process makes the object processing easier. On the other hand, larger image information slows the speed of the algorithm down. What is more, in this project colored background is used because of the fact that, it is not easy to determine the objects into the black and white background.
- Color Density : Since the object determination depends on the color of the objects, 24 bit color density is used.

5.1.1. Color Coordinates

The image is received in RGB coordinates, which represents all colors by combining different levels of red, green and blue colors, the problem with this color space is that under different lighting conditions, there will be a shift in all three coordinates, therefore in order to locate a color that shift should be located in all three coordinates, this is not an easy matter, and requires some help in the form of different color space named HSV, in this color space H holds the Hue, meaning the essence of the color, the other two coordinates stands for saturation and value, this is helpful because the H coordinate is hardly affected by different lighting conditions, and therefore in this project every pixel dealt with was first converted from RGB to HSV.

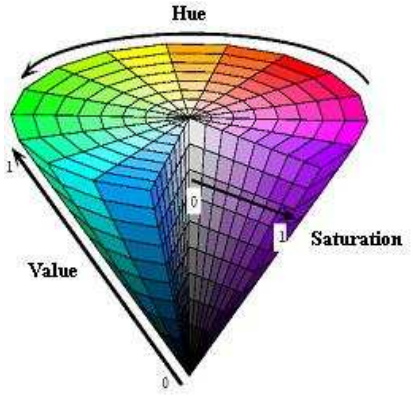


Figure 5.4. HSV color space [31]

5.2. Object Determination

The background picture is subtracted from the sampled picture. The difference matrix is being scanned pixel by pixel. Each pixel that is overlapped with the object properties is marked as VISITED. Thus, each pixel is visited once and this prevents to visit a pixel second time. If one of the pixel's components (R,G or B) exceeds a certain threshold this is saved in a list and then its neighbors are checked using the "Region growing" method. When a pixel enters the list, its absolute RGB values are transformed to one Hue value (using RGB to HSV transformation). Small lists are eliminated and each object's (list) attributes are saved: size, average hue and position of the center of mass. At the end of this stage, pixel lists of the objects are obtained. This data is always kept at the beginning of the program.

Program eliminates the objects which has a lower than 50 pixels. Program sees these objects as noise. Thereby, the field is cleaned from the noise effect. Hereafter, it is necessary to search thoroughly the pixel list that covers all the objects so that the color and the place of the pixels onto the objects can be found. Objects are the difference between the background picture and the sampled picture. If the difference on red color is more than 30 and the difference on green color is more than 25 and the difference on blue color is more than 30, then conditions are provided. Camera is less sensitive against green color.

5.3. Finding the Robot and the Target in the Object List

Robot and the target are seen as objects in the program. In every step, robot should know not only its position but also the target's position. This is done by grading the objects into the picture. Grading is made according to the color and size of the robot and the target.

Equation 5.1 represents the grading formula;

$$Grade = -A \times (Hue - KHue)^2 - B \times (Size - KSize)^2 \tag{5.1}$$

- $A = 0.1; B = 0.01 \Rightarrow$ These variables are used for the determination of the size and the color density. These variables provide a high grading for the objects. In this study, robot and the target are similar by size and by shape when they are viewed from above. That is why, the color density is being much more important for this project.
- Hue, Size $\Rightarrow$ These values give the color and the number of pixel of the robot.
- KHue, KSize $\Rightarrow$ These values give the color and the number of pixel of the target. (K=KNOWN)

After each object gets a grade, the object which has a highest grade is selected. In this project there are two objects that are looked for. One of them is yellow target (color: 55, size: 320), and the other one is green robot (color: 120, size: 320).

In the present case, the calculated grades are assigned to the robot and the target. Then, the red dots which are located at the backs of the robot and at that of the target are looked for into thorough their pixels. The coordinates of the robot and that of the target can be found with the help of these points; the centers of mass and the location of the red dots.

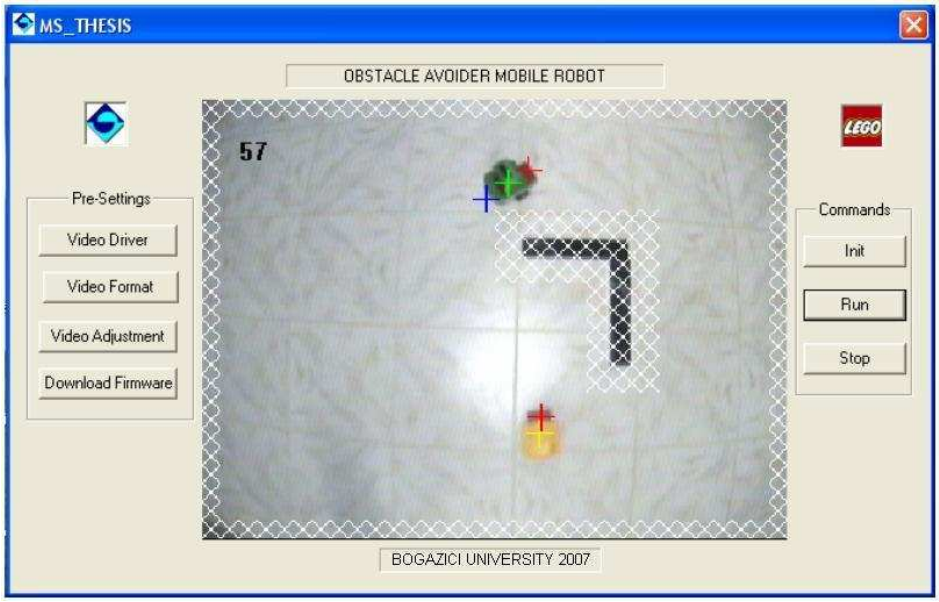


Figure 5.5. Robot and the target - decision of the next move "The Shortest Path Algorithm"

Yellow and Green crosses : Mark the target and the home base of the robot

Red Cross : Marks the red dot of the robot and the target

Blue cross : Marks the next move if "obstacle mode" is initiated

White X : Marks the obstacles

Counter : Indicates the sampling speed

5.4. Determination of the Next Move

The robot should know both its own coordinates and the target coordinates. Then, the algorithm draws an invisible line between the robot and the target in order to determine whether or not there is an obstacle between these two. If there is no obstacle between the robot and the target, then robot should go towards the target directly. If the line intersects an obstacle, then the algorithm determines the shortest

path between the robot and the target so that the robot could follow this path without hitting an obstacle. The algorithm should have;

1. Low resolution background space + obstacles. In the program, "**1**" refers to obstacle space (forbidden zone) and the "**0**" is free space where the robot can move freely.
2. Robot's coordinates
3. Target's coordinates

Algorithm: The angles determine the shortest path into the obstacle space and are calculated as in Figure 5.6.

```
SP(*MAT[width][hight], roverPos, targetPos)
{
      1) MAT[targetPos]=2
      2) SP recursive (*MAT, targetPos)
      3) Next cell find min( 9 neighbors of MAT[roverPos])
      4) If Next cell is North of roverPos return 0
      5) If Next cell is North East of roverPos return 45
      6) ...return 90
      7) ...return 135
      8) ...return 180
      9) ...return 225
      10) ...return 270
      11) ...return 315
}

SP_ recursive(*MAT,pos)
{
      Dirval ⇐ MAT[pos]+5
      DiagVal ⇐ MAT[pos]+7
      For each niePos which is a direct neighbor of pos
      {
```

Figure 5.6. The shortest path algorithm

If the value of the neighbors is higher and the target is still uncatched, then recursion is necessary.

```
    if (MAT[niePos]>Dirval) AND (Dirval<MaxVAL)
    {
        MAT[niePos]⇐ Dirval
        SP_ recursive(*MAT, niePos)
    }
For each niePos which is a diagonal neighbor of pos
{
    if (MAT[niePos]>Diagval) AND (Diagval<MaxVAL)
    {
        MAT[niePos]⇐ Diagval
        SP_ recursive(*MAT, niePos)
    }
}
```

Figure 5.7. The shortest path algorithm

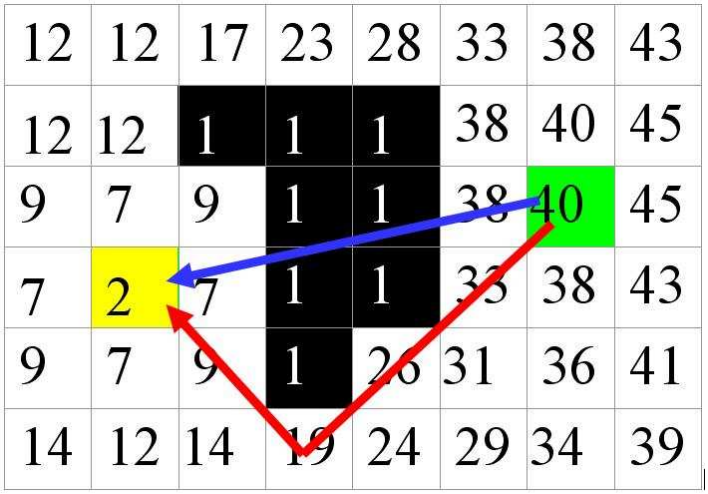


Figure 5.8. Yellow square represents the target while the green one represents the robot

The next move will be towards bottom left square.

5.5. Commands and Control

The robot should know its own position in order to determine the direction which it should go. Two different methods were examined to succeed this.

5.5.1. The Control Through the Existing and the Latest Coordinates

In this method, the coordinates of the two sequential movements' are kept and a vector is created between these two points. There are some pros and cons of this method. First of all, the shape of the robot is not important in this method. On the other hand, first move should be forward direction at high level so that the direction of the first move can be determined. This causes some troubles. Moreover, there should be long distance between the two sequential movements. This causes some serious troubles especially when the robot spins. In order to overcome this trouble, a lot of effort has to be made.

5.5.2. The Control Through the Red Point

A vector between the red point at the back of the robot and the center of the gravity can be created. Comparing the previous method, this method is much more convenient. Although, it has some disadvantages about the robot not having an aesthetical shape, this method is used in this study between these two methods.

α: The robot's direction is (in degrees) determined by the vector generated between the robot's center of mass and a red dot attached to its rear side. This angle is calculated by the Equation 5.2 where x1 and y1 are the coordinates of center of mass of the robot, while x0 and y0 are the coordinates of the center of mass of the red point of the robot. This angle varies between 0-360^0.

$$\alpha = \arctan\left(\left|\frac{y1 - y0}{x1 - x0}\right|\right) \tag{5.2}$$

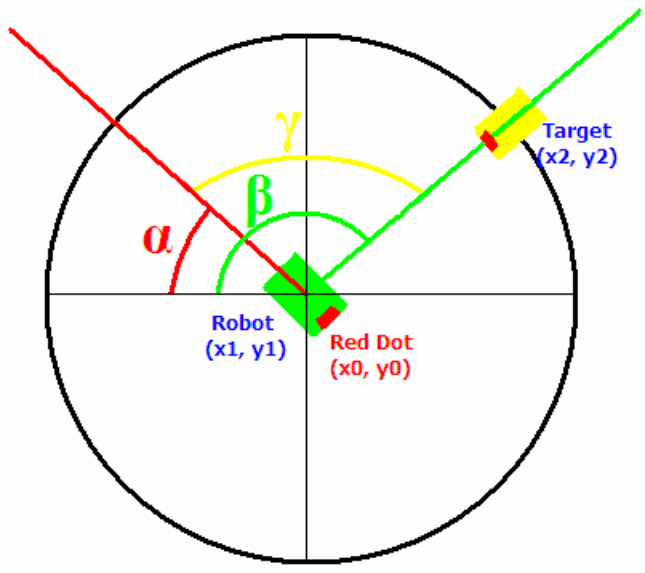


Figure 5.9. Calculations of directions

k is calculated in this way;

If $(y1 - y0)>0$ && $(x1 - x0)>0$ then k=0

If $(y1 - y0)>0$ && $(x1 - x0)<0$ then k=1

If $(y1 - y0)<0$ && $(x1 - x0)<0$ then k=2

If $(y1 - y0)<0$ && $(x1 - x0)>0$ then k=3

β: The target's direction is determined by the vector generated between the robot's current position and the target position. Equation 5.3 calculates the β angle, where x2 and y2 are the coordinates of the center of mass of the target.

$$\beta = \arctan\left(\left|\frac{y2 - y1}{x2 - x1}\right|\right) + 90k \tag{5.3}$$

$$\gamma = \alpha - \beta \tag{5.4}$$

Equation 5.4 gives the movement direction that is the difference between the two angles;

DIR represents the direction that the robot should go in order to reach the target.
If y>0 then DIR=LEFT
If y<0 then DIR=RIGHT

$$\gamma = |\gamma| \tag{5.5}$$

If $\gamma > 180$ then change DIR

$$\gamma = 360 - \gamma \tag{5.6}$$

The Azimuth angle between the robot and the target is calculated and it varies between 0-180^0. Hereafter, the only thing should done is to send the necessary control commands to the RCX according to this angle. The robot has two wheels and each wheel is connected to a separate motor. Each motor has 8 speed levels and the motors are connected to the different RCX outputs. At "0" motors don't move while at "8" they move with maximum speed. Three modes of movement for the robot are:

1. Forward
2. Curve left / right
3. Spin left / right

Table 5.1. Three mode of movement for the robot

Mode of Movement	Angle
Move Forward	Up to 10^o
Curve right/left	$10^o < \gamma < 45^o$
Spin right/left	$45^o < \gamma < 180^o$

In order to let the robot move forward, both motors' speed should be set to level 8. Actual speed values vary between 1-8. If one motor is set to 1 and the other motor is set to 8, this movement creates a chamber with a 70 mm diameter. One of the most important aspects is how to get a spin movement. Spin movement can be obtained

by means of setting the one motor forward while setting the other motor backward direction. If the motors' speeds are set to highest level, then precise spin movement can not be obtained. That is why spin movement is designed as one motor is set to zero speed while the other is set to the slowest speed. This provides a precise spin movement control.

5.6. Setting the Velocities

One of the most important aspects of this study is to determine the appropriate velocities for each movements. Any delay in the process can easily affect the whole control commands and some imperfection can occur. It is possible to get the optimal velocities with the help of following matters.

5.6.1. Shortening the Algorithm for Passing the Obstacles

The algorithm used in the program, is a recursive algorithm and this property causes time lag. In other words, it takes long time to converge. In order to prevent the loss of time, a limit is assigned to each pixel value. This limit refers to the length of the running track and the recursive algorithm stops beyond this limit. If this limit is exceeded, then the program stops automatically and the robot does not move anymore. Throughout the process, robot and the target coordinates are determined and then there is no need to search the coordinates of robot and the target anymore. The searching algorithm stops after finding the coordinates.

5.6.2. Cancelation of the New Signals

Phantom ActiveX module has such a program that it sends each command 5 times automatically. Although it provides better signal receiving property, the program stops and waits each time because of the time delay. As being a solution to this problem, repetition is canceled and only the new signals are sent to the RCX. Even if the signals are sent only once to the RCX, the IR Tower does this job without having a trouble. After making these improvements, 2 Hz signal rate is obtained. Continuous movement

can be accomplished by this velocity and this velocity is equal to the 13 cm/sec speed with the robot.

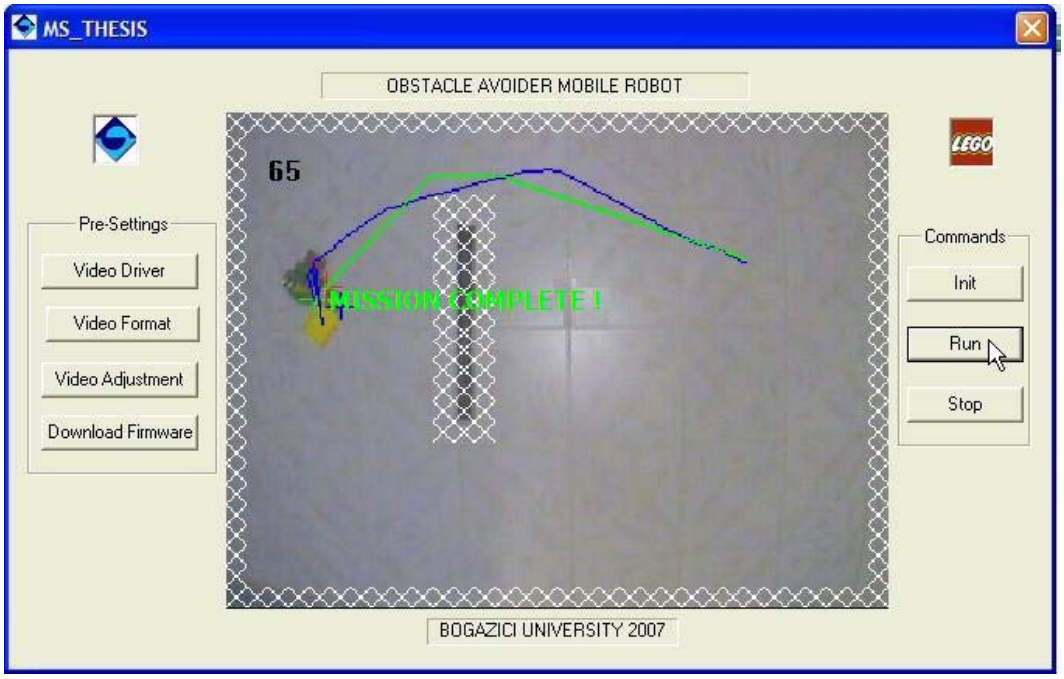
5.7. Communication

If there are suitable control commands, then it is possible to convert them into the numerical values. As it can be seen in the results section, these numerical values can be sent to the RCX thanks to the IR transmitter. As long as the control commands are available, this loop will continue.

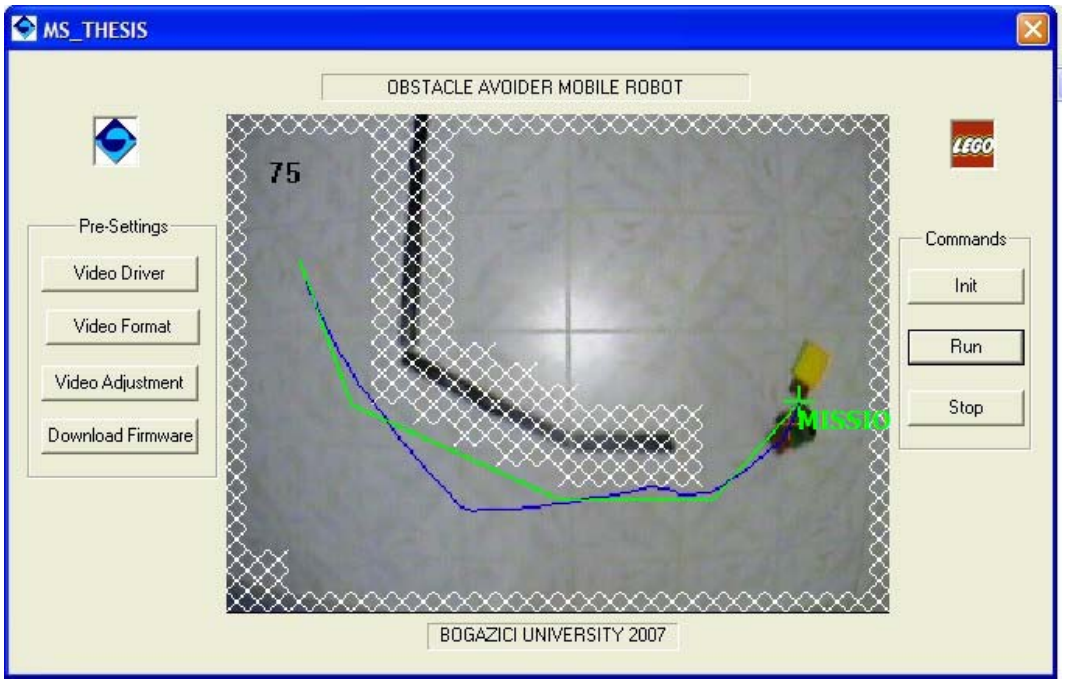
5.8. Catch

The algorithm can not recognize side by side objects and it sees these objects as one double sized object. When the robot reaches the target, the robot and the target are seen as one doubled sized object instead of being separate objects. This object is bigger than 700 pixel and its color is equal to the average value of the target's and the robot's colors.

Determination of the catching is easy discernible. When it happens, the robot stops, it gives us notice of catching by sound and "MISSION COMPLETE!" message appears on the computer screen. As it can be seen from the Figure 5.8 and Figure 5.9, the program also draws the blue colored real path that is already followed by the robot. Moreover, the green colored path the robot should follow, represents the shortest path, was drawn manually just before the execution of the program. By pressing the run button again, the program restarts.

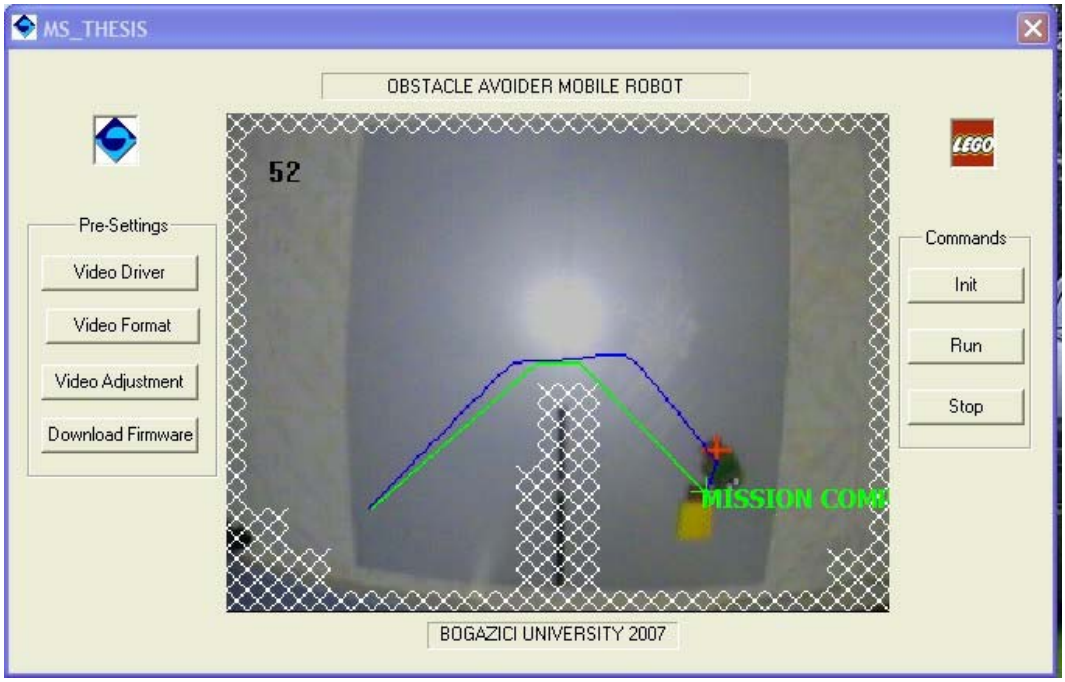


(a) Case I

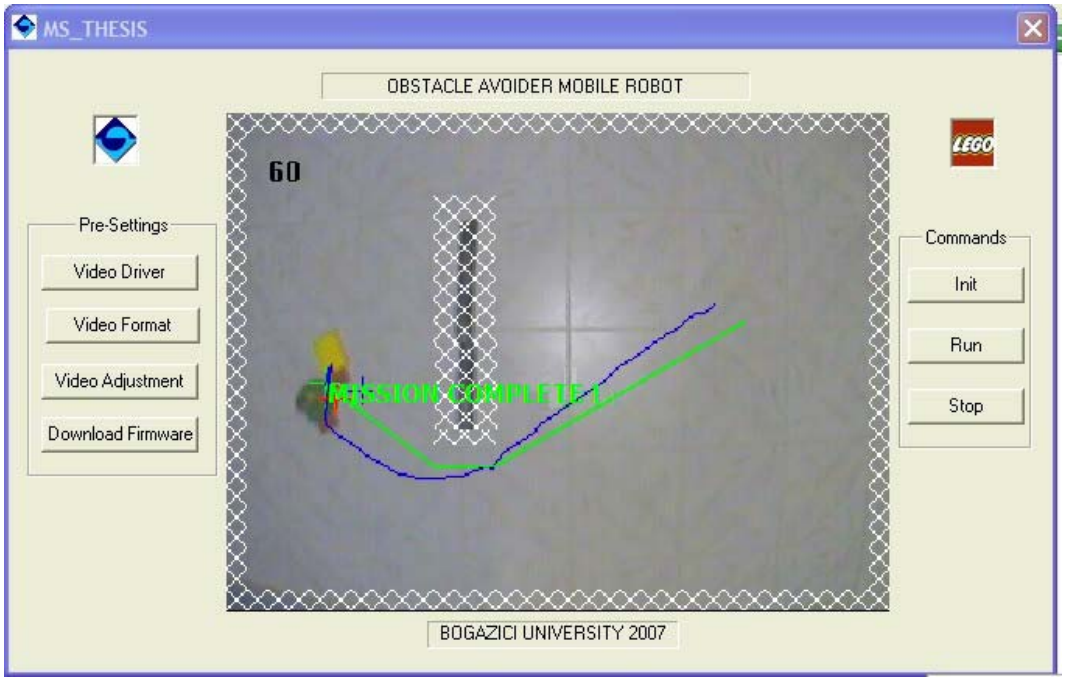


(b) Case II

Figure 5.10. Examples of real implementation



(a) Case III



(b) Case IV

Figure 5.11. Examples of real implementation

6. CONCLUSION

This thesis deals with the control of a Lego robot to reach a specified target in an environment with obstacles. In this study, seven different path planning algorithms were examined and these algorithms were compared according to their Matlab simulation results. Among these algorithms, VisBug algorithm was selected as the most suitable path planning algorithm and implemented to our Lego robot in this study. Moreover, program codes were compiled in Microsoft Visual C++ and a user interface was developed with this programming language. Matlab was used only for the simulations.

Although at the beginning of this study, it was proposed that an onboard camera would be used, this was changed to a dome type camera which sees the environment completely from above. Because all these seven path planning algorithms which are examined in this study, need to determine the exact position of the robot, obstacles and the target at the beginning, so that the robot could reach the target on the shortest path. In other words, if it is preferred to use one of these methods while proceeding to get from initial position to the final position, upfront knowledge of the environment is required. That is why bird's eye-view camera was used in this study.

It was possible to get reasonable results within this study. The system can be described as being robust against all noises and disturbances and it has high reliability. Even if some other objects are added to the workspace as disturbances, the program could easily determine the robot and the target according to their color and size properties.

On the other hand, despite these good results, the robot can sometimes hit the obstacles because of different lighting conditions and the problems in the obstacle avoidance algorithm. When the lighting conditions or the background of the workspace was changed, we had to recalibrate our camera and its color settings to get rid of unexpected program errors. As it was mentioned before, obstacle environment can

be extended and the physical collision can be avoided thanks to the Region Growing Method. Moreover, the robot sometimes goes into the forbidden zone. When the robot enters to the forbidden zone, it follows a zigzag course and it enters and then exits from this zone. The best solution to this problem was to control the next movement of the robot. If the pixel of the next move belonged to the forbidden zone, the robot had to be redirected to another way. This technique also has some disadvantages. If the free space that robot can move freely is narrow like a labyrinth, then it can have some problems. However, if there is no many obstacles into the space, then it works quite well.

There are some other methods dealing with the obstacle avoidance and, A* algorithm is the best one among these algorithms. A* algorithm is better than the algorithms that are examined in this study. It is also possible to add some sub algorithms, which calculate the direction that the robot should go and determine the better side to surpass the obstacles, to the A* algorithm.

Apart from these, a mobile target could be added to this system so that the robot could follow and reach this target with the same procedures. In order to do this, some codes had to be rewritten so that only one IR tower could send the necessary control commands to the RCX units of the robot and the target at the same time. Indeed, it was not tested to control two different RCX units at the same time. On the other hand, "**setvar**"control command of Phantom ActiveX modules could give this opportunity to us.

REFERENCES

1. Bujak, J. and G. Miroslav, "Mechanical State based modelling of control strategies and automatical code generation for mobile robots", *International Journal of Robotics Research*, Diploma Thesis, Wismar, 2003.

2. Latombe, J. C., "Motion Planning: A Journey of Robots, Molecules, Digital Actors and Other Artifacts", *International Journal of Robotics Research*, Vol 18, No.11, pp 1119-1128, 1999.

3. Schwartz, J. T. and M. Sharir, "On the Piano Movers' Problem: III Coordinating the Motion of Several Independent Bodies: the Special Case of Circular Bodies Moving Amidst Polygonal Barriers", *International Journal of Robotics Research*, Vol. 2, No.3, pp 46-75, 1983.

4. Joseph, J. and J. Pfeiffer, "A Rule-Based Visual Language for Small Robotic Applications", *Journal of Visual Languages and Computing*, Elsevier, Vol. 9, No.2, pp 127-150, 1998.

5. Nilsson, N. J., "A Mobile Automation: An Application of Artificial Intelligence Techniques", *In Proceedings 1st International Joint Conference on Artificial Intelligence*, pp 509-520, 1969.

6. Heger, N., A. Cypher and D. C. Smith, "Cocoa at the Visual Programming Challenge 1997 ", *Journal of Visual Languages and Computing*, Elsevier, Vol. 9, No.2, pp 151-169, 1998.

7. Lund, H. H., "Adaptive robotics in entertainment", *Applied Soft Computing Journal*, Elsevier, Vol. 1, No.1, pp 3-20, 2001.

8. Perez, T. L. and M. A. Wesley, "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles", *Communications of the ACM*, Vol 22, No.10, pp

560-570, 1979.

9. Lozano-Perez, T., "Spatial Planning: A Configuration Space Approach", *IEEE Transactions on Computers*, Vol C-32, No.2, pp 108-120, 1983.

10. Brooks R. A. and T. Lozano-Perez, "A Subdivision Algorithm in Configuration Space for Findpath with Rotation", *In Proceedings 8th International Joint Conference on Artificial Intelligence*, ICAI, pp 799-806, 1983.

11. Khatib, O., *Real-time Obstacle Avoidance for Manipulators and Mobile Robots*, International Journal of Robotics Research, Vol 5, No.1, pp 90-98, 1986.

12. Barraquand, J. and J. C. Latombe, "Robot Motion Planning: A Distributed Representation Approach", *International Journal of Robotics Research*, Vol 10, No.6, pp 628-649, 1991.

13. Kavraki, L., P. Svestka, J. C. Latombe and M. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces", *IEEE Transactions on Robotics and Automation*, Vol. 4, No.12, pp 566-580, 1996.

14. Barraquand J. and J. C. Latombe, "Nonholonomic Multibody Mobile Robots: Controllability and Motion Planning in the Presence of Obstacles", *Algorithmica*, Vol. 10, No.2-4, pp 121-155, 1993.

15. Laumond, J. P., P. E. Jacobs, M. Taix and R. M. Murray, "A Motion Planner for Nonholonomic Mobile Robots", *IEEE Transactions on Robotics and Automation*, Vol. 10, No.5, pp 577-593, 1994.

16. Svestka, P. and M. H. Overmars, "Motion Planning for Car-like Robots using a Probabilistic Learning Approach", *International Journal of Robotics Research*, Vol. 16, No.2, pp 119-143, 1995.

17. Sekhavat, S., P. Svestka, J. P. Laumond and M. Overmars, "Multi-level path planning for nonholonomic robots using semiholonomic subsystems", *International Jour-*

nal of Robotics Research, Vol. 17, No.8, pp 840-857, 1998.

18. Donald, B. R., P. G. Xavier, J. F. Canny and J. H. Reif, "Kinodynamic Motion Planning", *ACM*, Vol. 40, No.5, pp 1048-1066, 1993.

19. Choset, H., K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki and S. Thrun, "Principles of Robot Motion Theory, Algorithms, and Implementation", MIT Press, pp 1-164, 2005.

20. Schwartz, J. T. and M. Sharir, "On the piano mover's problem: II. General techniques for computing topological properties of real algebraic manifolds.", *Advances in Applied Mathematics*, pp 4:298-351, 1983.

21. Robotics Research Group, http://www.robots.ox.ac.uk/∼pnewman/Teaching/C4CourseResources/C4BMobile Robots.pdf, 2006.

22. Lumelsky, V. and A. Stepanov, "Path planning strategies for point mobile automation moving amidst unknown obstacles of arbitrary shape", *Algorithmica*, pp 403-430, 1987.

23. Kamon, I. E. R. and E. Rimon "A new range sensor based globally convergent navigation for mobile robots ", *IEEE Internatinal Conference On Robotics and Automation*, Minneapolis, MN, 1996.

24. The Computer Action Team, http://web.cecs.pdx.edu/∼mperkows/CLASS_479/S2006/lec014_MotionPlanning.pdf, 2006.

25. Jaillet, L., A. Yershova, S. M. Lavalle and T. Simeon, "Adaptive Tuning of the Sampling Domain for Dynamic Domain RRTs", *In IEEE RSJ Int. Conf. on Intelligent Robots and Systems*, pp 4086-4091, 2005.

26. Van Den Berg, J. P. and M. H. Overmars, "Using Workspace Information as a Guide to Non-Uniform Sampling in Probabilistic Roadmap Planners", pp 1055-1071, 2005.

27. Columbia University, www1.cs.columbia.edu/∼allen/S07/NOTES/Probabilistic-path.ppt, 2007.

28. Fogel, E. and D. Halperin, "Exact and Efficient Construction of Minkowski Sums of Convex Polyhedra with Applications", *In Proceedings 8th ALENEX*, 2006.

29. The RRT Page, http://msl.cs.uiuc.edu/rrt/, 2006.

30. Uppsala Universitet, http://www.it.uu.se/edu/course/homepage/realtid/dvnvht-06/ass2/rcx.gif, 2006.

31. The MathWorks Inc., http://www.mathworks.com/access/helpdesk/help/tooox/images/hsvcone.gif, 2007.