ROBOT PATH GENERATOR
Sreevani Pelala

GRADUATE PROJECT REPORT

Submitted to:
Department of Computing and Mathematical Sciences
Texas A&M University, Corpus Christi, TX.

Fall 2000

COMMITTEE MEMBERS:

R. Stephen Dannelly, PhD
Project Chairperson

Mario A. Garcia, PhD
Member

Carl Steidley, PhD
Member
# Table of Contents

i. List of figures ....................................................... iii

ii. Abstract ........................................................... iii

1. Introduction ....................................................... 01
   1.1 Background .................................................. 01
   1.2 Project Overview ............................................ 03
   1.3 This Document ............................................... 04

2. Capabilities of Robot ........................................... 05
   2.1 Saphira ....................................................... 05
   2.2 System Resources ............................................ 08

3. Best First Search Algorithm .................................... 10

4. Example of Best First Search Algorithm ......................... 15
   4.1 Route finding from Arad to Bucharest ..................... 15
   4.2 Navigation of robot .......................................... 20

5. Steps to complete the project .................................. 22
   5.1 Familiarizing with Saphira and pai Manual ............... 22
   5.2 Use of the simulator ........................................ 22
   5.3 Creation of the worlds ...................................... 23
   5.4 Understanding the ROBOT-GUI .............................. 28
6. Description of the algorithm .......................................................29
   6.1 How the code works ..............................................................29
   6.1 Major functions in the algorithm ...........................................43
   6.1 Testing the algorithm ...........................................................48

7. Results and Future Work .........................................................56

8. References ...............................................................................58

9. Source Code ................................................................................Appendix I

10. Pioneer World Specification files ............................................Appendix II
LIST OF FIGURES

1. Figure 1 – Client/Server system of Robot ...................................................07
2. Figure 2 - Interaction with path-generator ..................................................07
3. Figure 3a – Best First Search Algorithm .....................................................13
4. Figure 3b – Best First Search Algorithm .....................................................14
5. Figure 4- Map of Romania with road distances in km ...............................18
6. Figure 5 – Straight line distances in route Finding problem .......................19
7. Figure 6 – Best First Search Through Obstacles .......................................21
8. Figure 7 – Example world with point artifacts .........................................25
9. Figure 8 – Order of Boundaries in World Specification ............................26
10. Figure 9 – Example world with line artifacts ..........................................26
11. Figure 10 – Example world with three obstacles ...................................27
12. Figure 11 – Robot Path-Generator Flow Chart ........................................31
13. Figure 12– Robot Path-Generator Module CheckY ( ) Flow Chart ............32
14. Figure 13– Robot Path-Generator Module CheckX ( ) Flow Chart ..........33
15. Figure 14– Robot’s Path Example #1 ......................................................35
16. Figure 15– Robot’s Path Example #2 ......................................................36
17. Figure 16– Robot’s Path Example #3 ......................................................37
18. Figure 17– Robot’s Path Example #4 ......................................................39
19. Figure 18– Robot’s Path Example #5 ......................................................40
ABSTRACT

This project is the development of a “path finder” module for the navigation of a mobile robot using the “Best First Search” algorithm. This algorithm will be used to determine a path for a robot between two fixed points around a series of intermediate obstacles in a predefined external world.
1. INTRODUCTION

1.1 Background

A robot can be defined as a re-programmable multi-functional manipulator designed to move materials, parts, tools or specialized devices through various programmed motions for the performance of variety of tasks [10]. In other words, a robot is a force through intelligence where AI meets the real world.

Discoveries made during the last few decades in the field of artificial intelligence (AI) have allowed many different ways to emulate human processing of information. These discoveries have allowed the development of robots that can perform a variety of complex tasks e.g., open-heart surgery assistance. In assigning tasks to robots, much is dependent on the human operator decision making. This dependency is one of the highest priority research topics among the AI researchers whose goal is to obtain autonomous mobile robots [1].

An autonomous mobile robot is one capable of making a decision by itself based on a control algorithm which emulates human decision making [10]. As technology is advancing, autonomous robots are gaining more practical use in a variety of applications like manufacturing, construction-waste manipulation, space exploration, undersea tasks, assistance to the handicapped, and surgery.
In the direction of exploring various applications of robots, outstanding research is going on using a variety of robotic systems, which serve as versatile platforms for many purposes. Some of the innumerable uses of these valuable systems include navigation, error handling, visual perception and multiple-team operations. Some robotic systems are also used as an inexpensive base of prototyping platforms in product design groups like value-added resellers, engineering firms, government technology-transfer partners, and product-design groups. Some other applications where robotic systems are used are in delivery and collection, tracking and following, inspection and surveillance, tour guides, waiters and performers.

Robotic systems are quickly finding a place in university-level computer science and robotics courses. Also they are seen in high schools and elementary schools. Several courses at the university-level like C/C++ programming, client/server controls, sensors and effects and I/O controls, path planning and navigation, decision making and fuzzy logic, subsumptive architectures and multi-agent behaviors can be taught using these systems.

In view of conducting research in this aspect of technology which paves the path for innumerable benefits to mankind, the Department of Computing and Mathematical Sciences at Texas A &M University - Corpus Christi (TAMU-CC) is interested in research using the ActivMedia robotics system named ‘Pioneer’ which is a truly off-the-shelf, “plug-and-play” robot containing all the basic components for sensing and navigation in a real world environment including battery power, drive motors and wheels, position speed encoders, and integrated sensors and accessories, all managed via an onboard micro controller and mobile-robot server software. The
ActivMedia robotic systems are smaller, lower powered and cheaper enabling their application in areas not previously considered [4].

Currently, the Department of Computing and Mathematical Sciences at Texas A &M University - Corpus Christi (TAMU-CC) is interested in the controlled navigation of the Pioneer robot. Creating an optimal path for navigation of a robot in the presence of obstacles is the current issue under consideration. This project aims at generating a path for the robot navigation.

1.2 Project Overview:

This masters project, titled “Robot Path Generator” is the development of a “path finder” module for the navigation of a mobile robot using the “Best First Search” algorithm (Section 3 of this report). This algorithm is used to determine a path for a robot between two fixed points and a series of intermediate objects in a predefined world. This enables the navigation of a robot between two specified points. The successful completion of this module will provide a foundation for the implementation of a Simulated Annealing algorithm. The Simulated Annealing algorithm is to be used in place of the Best First Search Algorithm and the coupling of this algorithm with fuzzy logic will eventually lead to the finding of the shortest path for robot navigation [8].

The module has been developed using the C-language and the Saphira library functions and is executed using an existing graphical user interface. The algorithm used in this project is the “Best First Search” algorithm described in the later sections of this report.
1.3. This document

This document describes the graduate project – “Robot Path Generator”. Section I gives a brief overview of the problem under consideration describing the need for its implementation. Section II describes the capabilities of the Pioneer robot, a class of robots developed by SRI Inc. Section III gives an explanation of the Best First Search algorithm, a heuristic algorithm used in this project. Section IV explains the steps involved in the project. Section V gives examples to provide a clear understanding of the algorithm. Section VI gives an account of how the code works to enable the robot navigation. Section VII, Future Work and Conclusions, gives the current state of the project and the ways in which it could be expanded to find the optimal shortest path for the robot. The appendices I & II depict the Pioneer World file in the project and the source code listing.
2. CAPABILITIES OF ROBOT

The robot used in this project is the "Pioneer" robot, which is a versatile intelligent mobile robotic platform, built on the client-server model by SRI International Inc. An intelligent robot can see, understand, reason and act in a dynamic and uncertain world [10]. The robot acts as a server and uses "Saphira" resources. "Saphira" is a mobile client applications development environment. The robot server handles the low-level details of the sensors and drive management, such as, collecting range finding information from on-board sensors, maintaining individual wheel speeds, positioning, heading and so on and requires the client to guide it.

2.1 Saphira

The Saphira client system consists of four main components that are named based on the major function each performs. These functions are as follows:

1.) Communication Interface: This interface interacts with the robot and provides a protocol for communication with the control of a robot server by retrieving real-time, real-world operating data from the robot and sending back commands to control robot activities.

2.) Navigation Interface: This is the Saphira’s intermediate layer which supports higher-level functions for navigation control and sensor interpretation, and for integration of robot accessories (plugins) such as grippers, bumpers, day/night cameras, compasses, tilt position sensors etc.
3.) Behaviors Interface: At its upper levels, Saphira provides state-of-art, fuzzy-logic-based control behaviors and reactive planning systems, recognition systems, map-based navigation, and registration systems. Saphira uses fuzzy control rules for implementing and integrating rudimentary robot control programs called behaviors. It has several predefined behaviors such as Keep Off, Constant Velocity, Avoid Collision, Stop etc. This interface also provides the tools for defining behaviors. The behaviors compiler translates a simple fuzzy-control-rule syntax into C-language-based code that is included in the Saphira client.

4.) Graphical user interface: Saphira also provides a full-featured, command-level graphical user interface for interactive monitoring and manual control of both the Saphira client and the robot server accessories. This project, however, makes use of the GUI developed by Texas A&M University-Corpus Christi for its ease of understanding and operation.

The diagrammatic representation of the interaction between the robot server, the Saphira systems and the user can be interpreted as shown in the Figure 1 and Figure 2.
Figure 1

CLIENT / SERVER SYSTEM OF ROBOT.

Figure 2

INTERACTION WITH PATH GENERATOR.
2.2 System Resources:

Major resources of the Pioneer robotic system are described below:

**Saphira library:** This is a collection of Applications Programming Interface (API) functions and also the C-language based libraries that can be utilized to write individual modules. The Saphira API aids in the development and manipulation of the GUI and also provides functions for the monitoring and navigation of the robot.

**PAI Library:** Pioneer Application Interface (PAI) is a library of functions and definitions for the control of Pioneer mobile robots. PAI works in conjunction with the Saphira application development environment, collecting many of its low level robot control features into simple and more easily managed APIs.

**Robot Simulator:** The robot simulator enables easy testing and debugging of the programs. World models (abstractions of the real world with linear segments representing a vertical surface of corridors, hallways and objects in them) are made available in which the navigation of a virtual robot can be seen on the screen. Worlds can be created and loaded into the simulator at runtime with robot navigation mimicking exactly the robot, although certain limitations have been noticed while working with the robot. These are described in detail section 6.3 of the report.

**Sonars:** Numbered from 0 through 6, there are 7 on-board sonars, which assist in object detection. There is an optional Fast Track vision system on board the robot, but is not used in this project since the sonars seem to respond accurately for the obstacles used in this project.
Purposeful behaviors (e.g., going to a specific location) are dynamically blended with reactive behaviors (e.g., avoiding a moving obstacle) to provide smooth and compliant control of the vehicle as it navigates according to a strategic plan. The combination of fuzzy control, local perpetual space, and procedural reasoning technologies provides for flexible, robust control of autonomous vehicle operating in a dynamic environment.
3. BEST FIRST SEARCH ALGORITHM

The algorithm used in this project is the Best First Search (BFS) algorithm [2], which is a heuristic algorithm. The term “heuristics” means a rule of thumb or strategy one can use to solve a problem. One particular heuristic may not work alone, but it can guide problem solving so that one can obtain a solution. Better yet, a heuristic can lead to the invention of a more effective heuristic. Heuristics were probably invented when there were too many algorithms to choose for a particular result [5]. Later on it became a fundamental concept in artificial intelligence for solving many problems in which questions of optimality, questions of minimizing or maximizing were not at all important. In general, a heuristic may not be an optimal policy with respect to some criterion, but it is an approximate policy. The determination of these approximate policies depends on the structure of the process and how we wish to describe it.

The Best First Search algorithm is one such algorithm that is used in the current project to locate the next obstacle around which the robot moves on its way to the goal. This algorithm is a “goal-directed” and “knowledge-based” algorithm [2]. In other words, the start and end points are well defined before hand and also while performing the Best First Search, all the obstacles detected in the vicinity of the robot (as the robot turns from 0 through 180 degrees) are taken into consideration.

The robot gets a sonar reading for every 10 degrees of its angular displacement starting from zero degrees to 180 degrees. All these distances are sorted and the angle against the shortest
distance is chosen and the robot turns to that angle and moves around that obstacle. Hence the obstacle located at the shortest distance to the current position is selected always to move around. An evaluation function ‘f’, shown below determines the obstacle around which the robot turns. 

\[ f(n) = h \]  where ‘h’ is the shortest straight-line distance from the obstacle and n is the state, the current location of the obstacle. The obstacles are ordered such that the one located at the shortest distance from the current position (best evaluation) is traversed first. This strategy will be aiming to minimize the distance to reach the goal. The function sfOccBoxRet( ) [5] is used to get the distance of the obstacle from the robot with the help of sonars.

The Best First Search strategy prefers to move around the closest obstacle at each point, to reach the goal without considering whether this will be best in the long run. This algorithm will find a path quickly, but the path might not be optimal, as it is greedy at each state. For example, in Figure 3a, the robot could have moved towards the obstacle 1 and reached its goal with shorter displacement rather than moving around all the obstacles 2, 3, 4 and 5 and finally reaching the goal with larger displacement. But because it gets greedy about taking the path through 2, 3, 4 and 5 since \( x < y \) (it gets greedy to take the path through \( x \)) and ends up moving the additional horizontal distance of \( (a + b + c + d + e) \) as shown in the figure. This is justifiable as a better path if \( y \) is greater than \( (a + b + c + d + e) \) but in cases where \( y \) is less than \( (a + b + c + d + e) \) it certainly turns out to be a non-optimal path. Although it is not one of the goals of the project to move around so many obstacles, it turns out to be an important outcome which serves as a very useful feature since, such a behavior is desirable if the purpose of the application is to snoop around and create a map of the robot’s world.
The pseudo-code of the algorithm implemented is [2]:

Put the start node on the list
If the node is the goal
   End;
Else If the list is empty
   There is no solution;
Else:
   Select node from the list such that f(node) is minimum
   If the selected node is the goal
      End;
     Else:
       Expand selected node and add all the successors to the list.
Repeat.

The robot will be navigating between the two fixed and known points ‘S’ and ‘E’, which are the starting and the ending points respectively (Figure 3b). I₁, I₂, I₃,...I₉ are all intermediary obstacles through the course of the path, which are at predefined distances from the point S and every other point. The numbers on the arrows show the approximate straight-line distances (not drawn to scale) from the current location to the respective obstacle. At each state when the robot encounters a hindrance in its path, the distances from the current location to all the obstacles is calculated, and whichever is less, that obstacle is chosen for going around. Therefore, the direction of motion is determined by straight-line distance from the obstacle. This continues until the goal is reached as shown in Figure 3b.
BEST FIRST SEARCH ALGORITHM

Figure (3a)
Best First Search Algorithm
4. EXAMPLES OF BEST FIRST SEARCH ALGORITHM

4.1 Route finding from Arad to Bucharest

One of the simplest Best First Search strategies is to minimize the estimated cost to reach the goal. That is the node whose state is judged to be the closest to the goal state is always expanded first. For most problems, the cost of reaching the goal from a particular state can be estimated, but cannot be determined exactly [2]. When the nodes are ordered such that the one with the best evaluation expands first, the resulting strategy is called Best First (Greedy) Search.

The heuristic ‘h’ can be any function, but \( h(n) = 0 \) only if \( n \) is the goal. Heuristic functions are problem specific. A good heuristic function for route finding is a straight-line distance to the goal. That is, for example, \( h_{SLD} \) can be calculated by knowing the map coordinates of the cities in Romania(Figure 4) [2].

\[
f(n) = h_{SLD}
\]

\( h_{SLD} \) = straight line distance between \( n \) and the goal location. Also, \( h_{SLD} \) is only useful because a road from A to B usually tends to head in more or less the right direction [2]. This is the sort of extra information that allows heuristics to help in reducing the search cost[2].

Figure 5 shows the progress of the greedy search to find a path from Arad to Bucharest. With the straight-line distance heuristic, the first node to be expanded from Arad will be Sibiu, because it is closer to Bucharest than either Zerind or Timisoara. The next node to be expanded will be Fagaras, because it is closest. Fagaras in turn generates Bucharest, which is the goal. For this
problem the heuristic leads to minimal search cost. It finds a solution without ever expanding a node that is not on the solution path. However, it is not a perfectly optimal path. The path it found via Sibiu and Fagaras to Bucharest is 32 Kilometers longer than the path through Rimnicu Vilcea and Pitești. This path was not found because Fagaras is closer to Bucharest in straight-line distance than Rimnicu Vilcea. Hence, the path through Fagaras was expanded first.

The search tends to show the solutions quickly, although, as shown in this example, it does not always find the optimal solutions. That would take a more careful analysis of the long term options, not just the immediate best choice.

Also, this search is susceptible to false starts. If we consider the problem of getting from Lasi to Fagaras, the heuristic suggests that Neamț be expanded first, but it is a dead end. The correct solution is to go to first Vaslui - a step that is actually farther from the goal according to the heuristic and then continue to Urziceni, Bucharest, and Fagaras. Hence, in this case the heuristic causes unnecessary nodes to be expanded [Figure 4]. Furthermore, if we are not careful to detect the repeated states the solution will never be found - the search will oscillate between Neamț and Lasi.

Greedy search resembles the depth-first search in the way it prefers to follow a single path all the way to the goal, but will backup when it hits a dead end. It suffers from same defects as the depth first search - it is not optimal, and it is incomplete because it can start down an infinite path and never return to try other possibilities. The worst case complexity for greedy search retains all the
nodes in memory. Its space complexity can be reduced substantially [2]. The amount of the reduction depends on the particular problem and the quality of the heuristic function.
Map of Romania with road distances in Km and straight line distances to Bucharest [2]

Figure 4
SIBIU
h = 253

TIMISOARA
h = 329

ZERIND
h = 374

ARAD
h = 366

FAGARAS
h = 178

RIMNICU VILCEA
h = 193

OARDEA
h = 380

BUCURESTI
h = 0

STRAIGHT LINE DISTANCES IN THE ROUTE FINDING PROBLEM [2]

Figure 5
4.2 Navigation of Robot:

Figure 6 focuses on the behavior of the robot according to the Best First Search algorithm when it encounters obstacles in its navigation (a detailed description of the navigation is presented in Section 6 of the report). At each point when the robot encounters obstacles, it takes the shortest straight-line distance to reach the end point E. Therefore at each point if its sonars don’t detect anything, the robot progresses forward constantly checking for its goal coordinates simultaneously. If anything is seen by the sonars on its way, the algorithm calls the BFS algorithm which performs the same heuristic (the shortest straight-line distance from the obstacle) and moves around it and this continues until the end point is reached.

In the Figure 6, assuming S and E are the starting and ending points and I₁, I₂, I₃... I₆ as the obstacles in its path. From the point S, the robot gets the straight-line distance from the obstacle that is closest in distance to it and moves around it. So it moves around I₁ and from there according to the Best First Search algorithm, the shortest path is taken and the I₄ is reached. From I₄ it reaches the end point E. The robot moves around each obstacle before it determines the next obstacle to move around.
Best First Search through the Obstacles

Figure 6
5. STEPS TO COMPLETE THE PROJECT

5.1 Familiarizing with the “Saphira” Manual and “PAI” manual:

The Saphira and PAI manuals [6 & 7] that come with the Pioneer robot include all the information required to program the robot for any desired action. The manuals have been used as a good reference to become acquainted with the operation of the simulator, for the creation of the worlds and becoming familiar with the Saphira and PAI library functions.

5.2 Use of simulator:

The Saphira robot simulator is an executable file located in /usr/local/saphira/current/ver61/ of Saphira [6], by name ‘pioneer’. By running the command “pioneer” at the prompt, the simulator is invoked if the path is set correctly. The simulator also helps in constructing 2-D models of real or imagined environments called worlds. World models are abstractions of the real world, with linear segments representing the vertical surfaces of corridors, hallways, and objects in them.

The simulator provides a neat way of loading the desired world from the “/worlds” directory located in the “/usr/local/saphira/ver61/” of Saphira (by default) or any other directory. A world is any file that has a “.wld” extension and is in the specified format discussed in the section “5.3 Creation of the Worlds”. A desired world can be chosen by clicking the “load” button present on the Pioneer GUI. There is an option to shrink or increase the size of the world for the
case of viewing. The “robot-visible” menu item puts the robot on the screen at the position (x, y, and angle) defined in the world file by default although, the robot can be moved by mouse to any point in the simulator world. This makes the simulator a very useful tool in debugging the programs as it mimics the robot. There are however, certain limitations, which have been discussed in detail in section 6.3 of the report.

5.3 Creation of the Worlds:

Worlds can be created as desired for the purpose of the program. Worlds for the simulator are defined as a set of line segments using absolute or relative coordinates. Comment lines that are used in the world files begin with a semi colon. All other non-blank lines in the file describe the width and height of the world in millimeters. The simulator won’t draw lines outside these boundaries. A world boundary rectangle keeps the robot from running outside the world.

The worlds could contain either point or line artifacts. The format for the simple world that consists of a boundary of 4 walls with 4 points at a specified distance is as shown in Figure 7. These are called point artifacts. The width, and the height are specified as shown and each of the x, y coordinates are in millimeters. The next four lines of the world definition file define the world boundaries. Figure 8 depicts the order in which the coordinates of the boundaries are written while creating the world file.

The world file ‘world1.wld’ is the example of a simple world file with line artifacts that has been used in the project (Figure 9). The world definition of a line artifact involves specifying 4
coordinates, 2 for each end point of the line, instead of two coordinates as shown for the point artifact. The first two numbers are X and Y coordinates at the beginning of the line segment and the second two are the X and Y coordinates of the end of the line segment. The coordinate system for the world starts in the lower left, with positive Y-axis pointing to the north and positive X-axis pointing to the east (right hand side).

The comment lines are shown to start with semi-colon in the file. As usual, the first two lines define the width and height of the world. The following four lines define the sides of the world boundaries. The obstacles are defined as box objects. The obstacles in this projects are restricted to rectangular (a square being considered as a rectangle with equal sides) box objects.

A box object is created with four lines of code. Each line defines a side of the rectangle. The four numbers in the line represent X, Y coordinates of the starting and ending points of the side of the rectangle. The order of defining the lines of the rectangle is as shown in the Figure 8. Sides 1, 2, 3, and 4 are created in that order. In other words, the X and Y coordinates of ab, ac, db, and dc create the rectangle. The coordinates in the world file are with respect to the global axis and hence they are calculated with respect to the origin. The last line of the file defines the position of the robot. The X and Y coordinates of the robot and the angle (theta) of the robot with respect to the global axis. In this file the robot is pointing to zero degrees and hence it faces towards east.

Figure 10 is the example of a world definition file with three obstacles.
;;; Fragment of a simplified world

width 10000
height 5000

; The first two numbers represent the X and Y coordinates of the starting point
; on the line segment and last two numbers represent the X and Y coordinates
; of the ending point of the line segment.

0 0 0 5000 ; World frontiers
0 0 10000 0
10000 5000 0 5000
10000 5000 10000 0

;first point (X, Y coordinates respectively.)
1100 1100

;second point (X, Y coordinates respectively.)
2600 2600

;third point (X, Y coordinates respectively.)
2600 3700

;fourth point (X, Y coordinates respectively.)
2800 3400

;position of robot (X, Y coordinates and angle respectively.)
position 0 0 0

EXAMPLE WORLD WITH POINT ARTIFACTS

Figure 7
ORDER OF BOUNDARIES IN WORLD SPECIFICATION
Figure 8

;;; Fragment of a simplified world

width 10000
height 10000

; Each line shows X, Y coordinates of the
; starting point followed by the X and Y coordinates of the
; ending point of the line segment respectively

0 0 0 10000 ; World frontiers
0 0 10000 0
10000 10000 0 10000
10000 10000 10000 0

2700 7600 5600 7600
2700 7600 2700 8900
5600 8900 5600 7600
5600 8900 2700 8900

position 5000 5000 0

EXAMPLE WORLD WITH LINE ARTIFACTS
Figure 9
; An example of a world with three box obstacles.

width 10000
height 10000

; Each line shows X, Y coordinates of the starting point followed by the X and Y coordinates of the ending point of the line segment respectively

  0 0 0 10000 ; World frontiers
  0 0 10000 0
  10000 10000 0 10000
  10000 10000 10000 0

; first obstacle

  6200 1800 7600 1800
  6200 1800 6200 2700
  7600 2700 7600 1800
  7600 2700 6200 2700

; second obstacle

  4400 3800 6000 3800
  4400 3800 4400 4900
  6000 4900 6000 3800
  6000 4900 4400 4900

; third obstacle

  1500 6500 3900 6500
  1500 6500 1500 8200
  3900 8200 3900 6500
  3900 8200 1500 8200

position 7300 800 0

EXAMPLE WORLD WITH THREE OBSTACLES

Figure 10
5.4 Understanding the ‘ROBOT - GUI’:

The path generator module has been added as a callback function (i.e., the function gets called on clicking the 'Start/Stop Navigate' widget in the Robot-GUI) in the robot GUI that has been created by a group of undergraduate students at Texas A & M University-Corpus Christi. The GUI consists of a neat environment to facilitate the preliminary movements of the robot like moving forward, backward, and making right and left turns using the appropriate widgets that call the “Saphira” and “PAI” library functions. The GUI has been created in X- windows on LINUX using the X-Motif library functions.

Some of the various widgets that the GUI is equipped with are the ones that aid in setting the speed, setting the heading, changing the tilt and pan values of the camera, Gripper opening and closing, and the preliminary help for the use of GUI. Also the GUI can initialize and set the sonar labels and also update them every 50 ms. It also displays the updated speed and location (x, y, th) of the robot in motion every 500 ms. Also, a module that makes the robot run and an other that makes the robot follow behind any object that it aims at have been implemented before by Chris Bono, an undergraduate student at Texas A & M University, Corpus Christi under the direction of Dr. Dannelly.

capi.h, saphira.h, pai.h files have been used as include files. The pai.h and saphira.h provide all the necessary functions used in the robot navigation while capi.h has been used where there is a need for slight modification of the library functions. The use of capi.h avoids making direct changes to the files saphira.h and pai.h.
6. **Description of the algorithm**

6.1 **How the code works:**

The algorithm aids in automated navigation of the robot from the starting point to the goal through a series of obstacles without colliding with the obstacles. While doing so, it will choose the obstacle that is closest to its current position (Best First Search) to move around. Traversing obstacles between the start point and the goal provides an effective means of investigating and mapping the robot’s environment.

The flow chart for the algorithm is shown in Figures 11, 12 and 13. The robot’s global coordinates at its starting position (x, y, angle) should always be (0,0,0). The angle ‘0’ means that the robot points to east or along the positive X-axis. Hence the robot at its starting position always points towards east. Therefore, in the execution of the algorithm, the robot first changes its angle to point to the north to progress in the positive Y direction. According to the algorithm, the robot always tends to progress in the positive Y direction first. (Figure 21). After it reaches its goal on Y-axis, it checks its X coordinate and progresses in the direction of the goal on X-axis.

If the robot encounters any obstacles in its way, it passes the obstacle either by moving around the obstacle (Figure 19, step 3) or just passing (Figure 22, step 13) by the obstacle depending upon the distance and orientation of the robot with respect to the obstacle. The robot performs a search for the next available obstacle before it progresses in the Y-direction. The search here is
the Best First Search since the robot turns at increments of 10 degrees each time getting the
distance of the obstacles in its vicinity. It points to the obstacle that is nearest to its location and
Robot Path Generator Module

Figure 11
While (!goalY())

If (!goalY())

aim()

If (!goalY())

Move_around()

If (!goal Y())

Get Nearest()

If (!goal Y())

CheckX()

Robot Path Generator Module CheckY ()
POINT STRAIGHT()

If (sfRobot.ax) > goal X-coordinate

If Forward Okay()

Get_around_X_g_0()

PaiMoveRobot(50)
If (!goal()) || (!ForwardOkay())
Break;

else

If sfRobot.ax < goal X-coordinate

If ForwardOkay()

PaiMoveRobot(50)
if (!goal()) || (! ForwardOkay())
break;

Get_around_X_g_0()

If sfRobot.ax == goal X-coordinate

GOAL REACHED.

Robot Path Generator Module - CheckX()
tries to pass around the obstacle. This is based on the heuristic that at each position of the robot it tries to take the shortest path by progressing towards the obstacle nearest to it.

The robot also keeps track of the Y coordinate when it is trying to pass the obstacle. When the Y coordinate coincides with the Y coordinate of the goal with a buffer of 200 mm (Y coordinate +/- 200 mm) the robot begins to check for the X coordinate. If the X coordinate of the goal is greater than the current position of the robot, it proceeds towards the west until it reaches the goal. If it is lesser than the current position of the robot, it moves towards the east to reach its goal X. This algorithm helps in finding the exact goal of the robot with certain restrictions like deviation from the goal, discussed below.

As long as the path is smooth without obstacles on its way to the Goal X, the robot reaches its goal with higher precision. If there happens to be obstacles in the path on its way to reach the X coordinate of goal G, it could bring a considerable deviation in the path of the robot from its Y coordinate as shown in the example paths (Figures 14, 15, 20, 16, 21, 22 etc). In some instances a deviation is inevitable since, if the goal happens to be a point inside the obstacle, it cannot be reached (Figure 14). In other situations (Figures 15, 16), a possible solution to refining this deviation would be integrating the algorithm with mapping wherein the robot is actually aware of its goal at a certain distance so that it could change its path and head towards the goal.

While heading towards its goal X the robot moves straight as long as there are no obstacles in its path. But once it encounters an obstacle, it tries to move around it rather than stop at that obstacle. Hence it moves around the obstacle and ends up at the correct X-location, which causes
INDEX:
G ---- Goal position
R ---- Robot's final position.

○ - Robot
d --- deviation from the goal and the final position of the robot.

ROBOT PATH
EXAMPLE #1
Figure 14
INDEX:
G----Goal position
R-----Robot's final position.

- Robot
d --- distance between the goal and the final position of the robot.

ROBOT PATH
EXAMPLE #2
Figure 15

X-AXIS