AM77: Search Algorithms

(Final Year Project)

Submitted by:

Tan Shing Yenn
(U006863M11)

Department of
Mechanical Engineering

In partial fulfillment of the
requirements for the Degree of
Bachelor of Engineering

Session 2003/04

Summary

Two search algorithms (the wall-following search and the “most-open-area” search) have been designed, tested, compared and analysed both in simulations and hardware experiments.

Dangerous search tasks such as bomb and chemical searches put human lives seriously at risk and it is imperative to utilise artificial intelligence and robotics to help us in these real-life situations. The search methods presented here have shown that search ideas can be incorporated into robotics and lay down the validations and foundations necessary for practical usage.

In these two algorithms, an autonomous mobile robot is programmed using C++ object-oriented programming language and relevant software interfaces to search for a light target source (a fluorescent light bulb) in a dark room. The search is done in a flat or office-like layout, which has rooms and door openings and it ends when the robot comes to a complete stop within a stipulated distance away from the light source.

Simulations are done with a robot control device software, *Player & Stage* and they provide the necessary testing platforms of the algorithms before they are fully implemented on the actual robot hardware. The hardware experiments are conducted using a simple servo-driven robot, equipped with only sonar and photoelectric light sensors and a magnetic compass. The factors used to compare the efficiencies of the two algorithms are the time taken to complete the search tasks as well as their success rates.

Final results show that the wall-following search is more efficient in terms of success rates compared to the “most-open-area” search. The latter search method though, provide a faster search method to complete the search in the majority of the cases.

Acknowledgement

I wish to first express my gratitude to my supervisor, Associate Professor Gerard Leng, National University of Singapore, for his kind supervision and mentorship during the entire length of this project. Special thanks must also go to Mr New Ai Peng from DSO National Laboratories for his assistance with the Player and Stage software. I would also like to thank Amy and Priscilla, technical staff of Dynamics lab for their assistance with the equipment and colleagues like Chee Kong, Wee Kiat, Yee Leong, Han Yong, Vincent, Stephen and Kwang Chern for their valuable inputs throughout this project. I also wish to thank my parents who have been very supportive and understanding throughout these past eight months. Last but not least, I would like to thank God whose grace has helped me pull through many difficulties encountered during the entire length of the project and who has been my inspiration.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Purpose of project

The main purpose behind this project is to compare and test the efficiencies of two search algorithms using a single autonomous robot. The mobile robot is expected to search for a light target source in a given area with an unknown but realistic layout that likens to that of a typical office or apartment flat environment.

1.2 Real-life applications of search algorithms

Searching for bombs, gas leakages, toxic chemicals or even human casualties in hostile environments are dangerous and life-threatening tasks. Nanotechnology has also supplied the possibility of micro bombs and spy gadgets that are almost impossible to detect using human senses. Hence, autonomous robotics provides excellent platforms and solutions to perform such searches without having to risk human lives.

The birth of artificial intelligence and mobile robotics has given us important foundations to utilise such scientific advances to help perform unmanned and autonomous search tasks. Many advanced ideas have been incorporated into search algorithms to extend their scope of usage. This report illustrates the powerful combination of such current technologies and the latest programming architecture in the development of two useful search algorithms. The value of these search algorithms is highly enhanced through their practicality and expandability values, as they are developed to perform in an actual office or flat-like human-dwelling environment.

1.3 Scope of project

This project required an extensive research into realm of autonomous robotics and search methods. Two different search algorithms (the wall-following and "most-open-area" search) went through a carefully planned designing stage and were implemented within given constraints of time and resources.

The types of intelligence imparted to the robot are very much related to the combination of the hierarchical and the reactive paradigms mentioned by Robin R. Murphy in his book "Introduction to AI robotics" [9]. Figure 1.1 below illustrates the thinking loop of the hierarchical paradigm and figure 1.2 depicts that of the reactive paradigm.

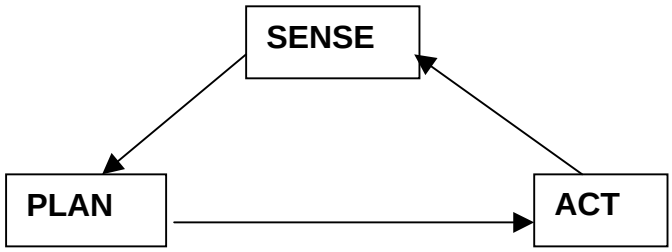


Figure 1.1: The hierarchical paradigm

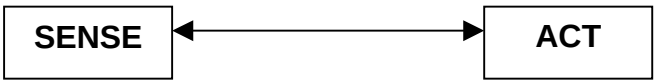


Figure 1.2: The reactive paradigm

It was consequential to learn the functions of the Linux Operating System as a requirement to create and run the simulations. The robot architecture software, *Player & Stage* (please refer to appendix 9.1 for more information), which is used in this project had to be mastered with a proper understanding on how to make use of an advanced robot device server to develop the

algorithms. Once the search algorithms were validated and tested in the simulations, the algorithms are subsequently implemented using C++ object-oriented programming on actual hardware equipment. On the hardware experimental side, it was salient to first develop a good knowledge on the mechanical functions of the robot mechanisms as well as sensor techniques. Lastly, many trial runs were necessary to fully test the algorithms on the actual robot before the full real-time experiments were carried out.

In this project, the two search algorithms are executed without the use of odometry. Odometry related techniques require very high level of accuracy and performance from the robot's mechanisms and sensors, which are not viable in this project. Problems such as wheel slippages and sensing inaccuracies are common and highly complex correction methods have to be used in order to rectify such problems.

1.4 Outline of thesis

The Literature Review chapter follows after this chapter and it serves to summarise important related works by researchers in the field of autonomous robotics and to link their researches to this project. Chapter 3 shows the design process of the search algorithms and describes the usage of various functions in each of the algorithms. The report then focuses on developing the search algorithms using the *Player & Stage* software in chapter 4, which includes test results and analysis. This report provides the results of hardware tests and analysis chapter 5. Chapter 6 gives a conclusion on the simulation and hardware test results and the characteristics of the two search algorithms.

In addition, there is also a chapter following the conclusion section regarding further discussion and recommendations in chapter 7. Appendices can be found in chapter 9.

Chapter 2: Literature Review

2.1 Brief introduction

For a mobile robot to perform search algorithms, it must be intelligent enough to navigate the environment, avoid obstacles and finally locating the search target with the help of adequate sensors. Many relevant works regarding robot autonomy have been studied and the usefulness of some of the ideas could be tapped and combined in an effective way to develop successful search algorithms.

2.2 Relevant works by researchers

Keith L. Doty and Steven Louis Seed [1] presented a search idea using a small, low-cost autonomous mobile robot with the use of mapping to identify landmarks. However, they found that such odometry-related search using a mobile robot equipped with simple sonar sensors would result in substantial errors and problems because of common problems such as wheel slippages and sensing inaccuracies.

Brian Yamauchi [2] introduced an exploration method using concepts of frontiers. Inherent in his research is also introduction of using laser-limit sonars to more accurately create evidence grids. This idea requires the robot to be able to detect frontiers in the updated evidence grid and attempt to visit the nearest unvisited frontier.

Some other researchers have already developed a strategy of wall following using angle information measured by an ultrasonic transducer [4]. Thus the

robot is able to turn accurately at convex and concave corners by a sufficiently fast measurement cycle compared to the speed of the robot, which allows the feedback intelligence to work purposefully. This is done by making use of the specular reflection ability of ultrasound and examination of changes in the peak echo frequencies. This research was experimented using a highly advanced mobile robot equipped with power steering and odometry mechanisms and a high-end transducer to measure the bearing angles accurately.

A philosophy of using collective features of the known environment to form a map [5] has also been researched into. Exploration algorithms using points of interests as links to search for goals, integrated with obstacle avoidance, together with local and global path planning algorithms are developed to determine the robot's trajectory. Such feature-based exploration is independent of the kind of sensing employed. However, their idea requires the assumption that their robot is equipped with as SLAM algorithm that is capable of consistent mapping and localisation. It should be noted that this is not possible to be performed with simply a mobile robot equipped with mediocre sonar sensors and no odometry capabilities.

A recent method to explore unknown environment without the use of GPS has been proposed by Maxim A. Batalin and Gaurav S. Sukhatme [6]. The main idea behind this approach is to use beacon markers, which are dropped off by the robot during navigation in the environment and each marker acts as a local signpost telling the robot which direction to go next.

Another new method of recursive autonomous exploration using the “next best view” approached was also recently developed. By assessing free space in an environment, multi robots [7] are able to share information and operate independently to explore regions of interests. This is done by making use of an inverse sensor model that identifies regions in space where a new sensor reading has maximal utility. In order for this method to work, accurate occupancy maps must be built and hence the sensors must be able to define objects distinctly.

2.3 Relating researched works to this project

This project makes use of a simple robot platform with no differential drive mechanism. Any approach using odometry would introduce many problems caused by wheel slippages and different servo speeds. The robot’s senses come from mediocre sonar and light sensors, which can be unpredictable at times and thus approaches, which rely on sonar sensing to identify landmarks or frontiers should be avoided also. However, what can be extracted from these researches is their general idea behind robot search and navigation techniques.

In linking their research ideas to this project’s platform, [4] provided the basis of a reliable search technique of wall following, taking note of convex and concave corners. [1] presented the possibility of using a low-cost robot for navigation and thus injected confidence into this project also. The first search algorithm (wall-following) therefore introduces these ideas on a much simpler robot platform.

Works by [7] supplied the main motivation to develop a search technique, which calls for the robot to assess the “most-open” area and move into the area to search. Again, this project’s second algorithm (“most-open-area” search) is developed without the use of their inverse sensor method or the complex process of mapping, which requires odometry.

Chapter 3: Designing the search algorithms

3.1 The motivation

The main source of motivation in the design of these two algorithms is the practical value of such search methods. It is important that these techniques can be applied in the most common environment in everyday life situation. Therefore a flat or office-like layout is chosen as the purposeful environment. Equally as salient is the contrast between the two search algorithms as the motivation to compare and analyse the two methods is lost if they use similar approaches. Hence, while designing the algorithms, there must be ample considerations from the motivating factors mentioned. The design flow chart shown in appendix 9.2 illustrates the making process of the search algorithms.

3.2 Assumptions

To illustrate the two search algorithms clearly, there are some assumptions made both in simulations and the hardware experiments. Therefore, in this project, the main assumptions made are:

a) All turning angles are either 90 or 180 degrees
b) Only walls (wooden planks of about 0.4m high) act as obstacles and there are no other obstacles lying around in the layout
c) The doors (entrances and exits) follow closely to real-life size, which are about 80 -100cm wide
d) The size of the mobile robot is approximately 4 - 5 times less than the width of the doors
e) There are no entrances and exits along the external walls

A simple diagram of what a possible layout look like is depicted in figure 3.1. This layout encompasses the basic features of a possible flat or office-like environment with connected rooms and door openings.

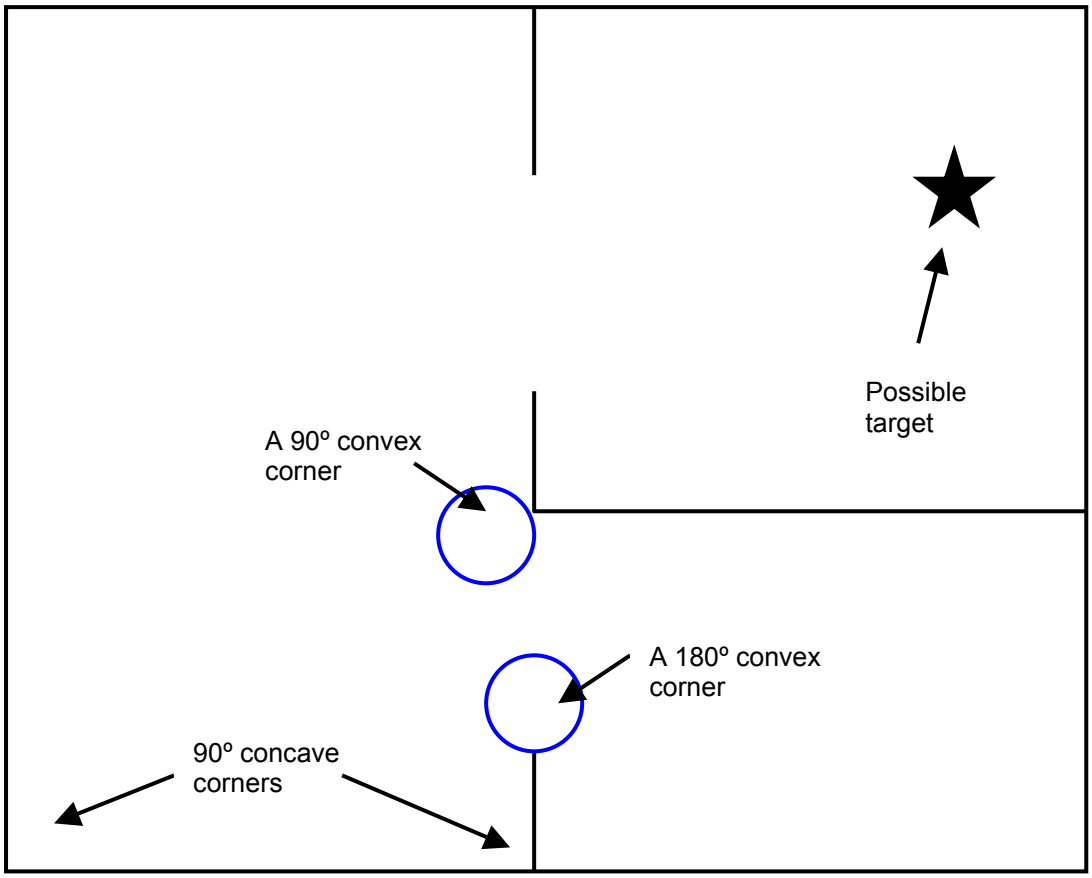


Figure 3.1: A simple illustrated floor plan with 3 types of turning angles

3.3 Search Algorithm 1: The wall-following search

A wall-following search is very useful in flat-like environments where walls act as main obstacles and separators of rooms. Therefore, if the robot is able to follow the wall say on its right side, it will be able to move into every room in such a layout. Hence, this search method runs on the idea that the robot keeps near to the wall while navigating in the environment and its main pseudo-code loop looks like this: -

```
while (start flag)
{       if obstacles are blocking the robot
        {
                Go into the obstacle avoidance function (3.3.1)
                if no wall near right side, then call the
                {
                        find wall function (3.3.2)
                        followed by wall follow function
                }
                        else call the wall follow function(3.3.3)
        }
        else
        {
                if no wall near right side, then
                {
                        call the find wall function (3.3.2)
                        and then call the wall follow function (3.3.3)
                }
                else call the wall follow function (3.3.3)
        }
}
```

3.3.1 Obstacle avoidance function

This is a simple function, which allows the robot to turn out of a blockage situation. The robot will turn on the spot until it senses that its front is clear enough to forward movement, which is illustrated in figures 3.2 and 3.3.

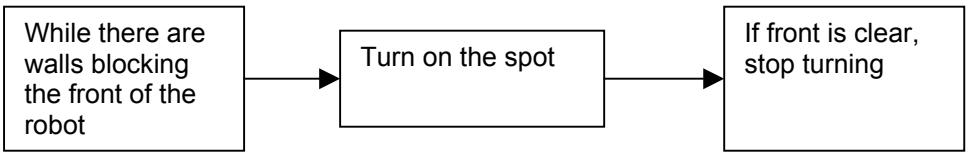


Figure 3.2: Obstacle avoidance function

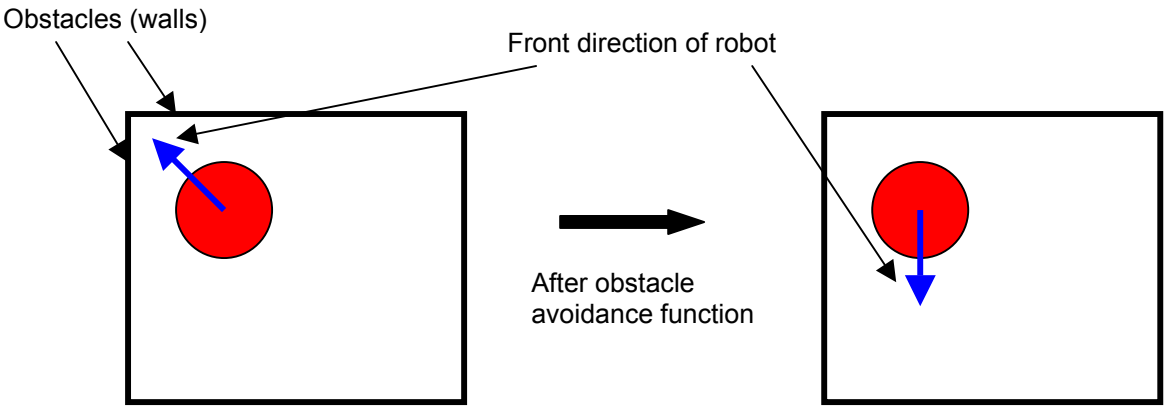


Figure 3.3: Picture illustration of the obstacle avoidance function

3.3.2 Find wall function

Basically, this function shown in figures 3.4 and 3.5 allows the robot move to find a wall when it cannot sense a wall on its right side. When a wall has been found, the robot will try to parallelise against the wall to prepare itself to follow the wall.

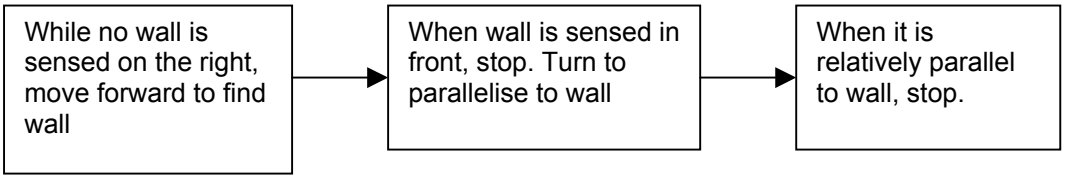


Figure 3.4: Find wall function

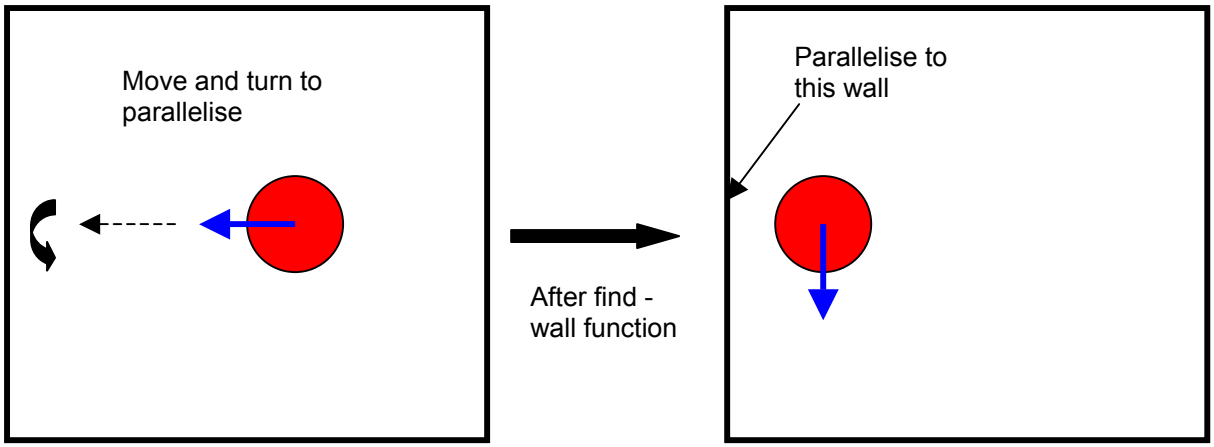


Figure 3.5: Picture illustration of the find wall function

3.3.3 Wall-follow function

As depicted in figures 3.6, 3.7 and 3.8, when the robot sees the wall on its right, it drives straight. While the wall is on its right, if it veers too close to the wall, it will pick this up and turn away, and if it veers too far away from the wall, it will sense it and veers back in. If while following the wall, the robot encounters a 90º concave corner, it will negotiate the corner by turning on the spot and start following wall again when completes the turn. Likewise, it does the same when a 90º or 180º convex corner is detected. As the robot performs the wall follow function, it is also scanning for the light source in a read-think-act loop. In the hardware robot, five light sensors are chosen to be used to detect any large intensity readings, which signifies a light source is nearby. These five light sensors are able to cover a minimum of 180 degrees range span relative to its front direction. When a large enough intensity reading is recorded, it goes into the search function.

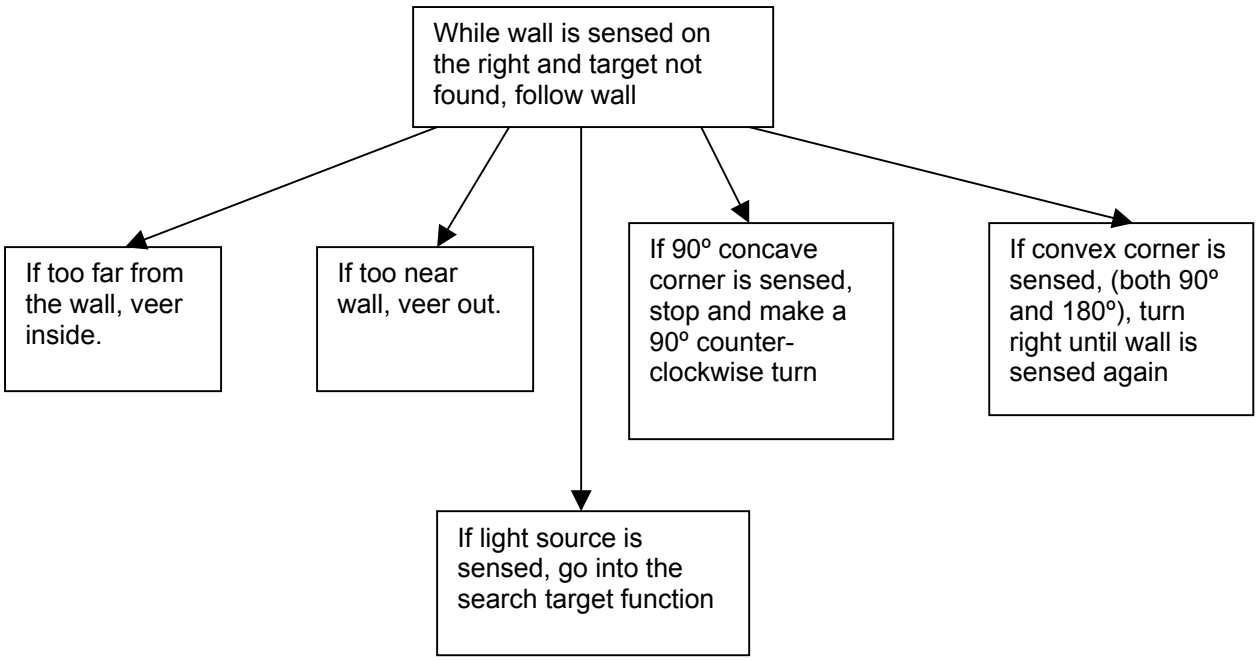


Figure 3.6: Wall-follow function

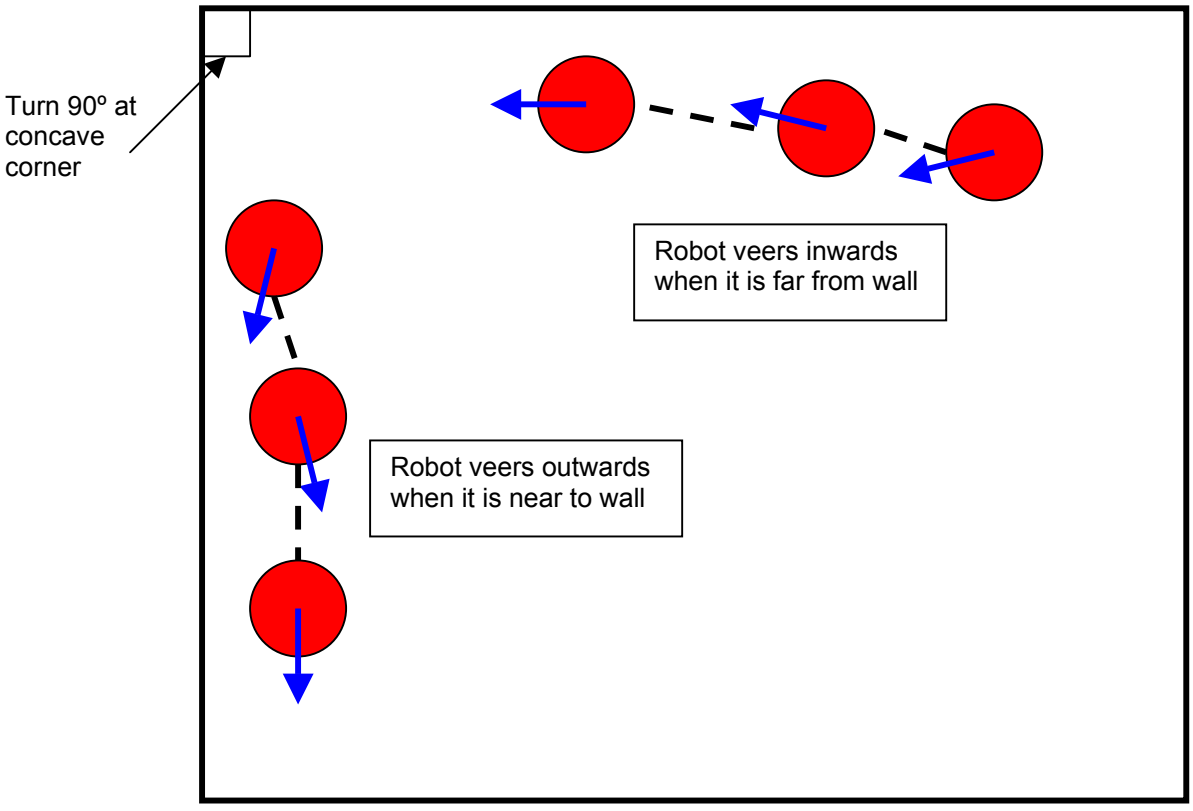


Figure 3.7: Picture illustration of the wall-follow function

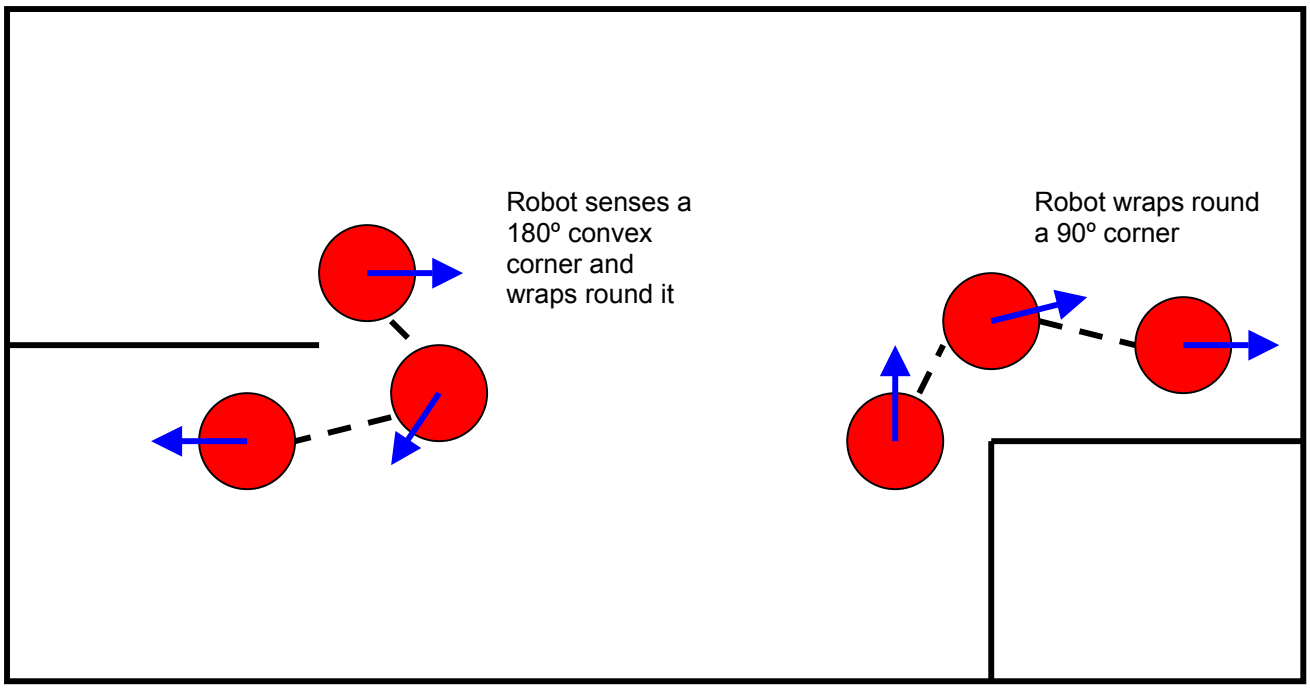


Figure 3.8: Another picture illustration of the wall-follow function

3.3.4 Search function

In the search function illustrated in figures 3.9, 3.10 and 3.11, the robot continues to keep track of the high intensity reading move towards the light source while moving towards the light source. It will try to move in a straight line towards the light source and if the robot starts to veer away from it, a change in its intensity readings will be detected and it will adjust back (either veer left or right) until the light source is more or less within its frontal view once more. It also has obstacle avoidance ability such that it does not collide with the edges of the walls and it serves to guide the robot safely to the target. Finally, the robot halts within 30cm from the light source and the algorithm ends.

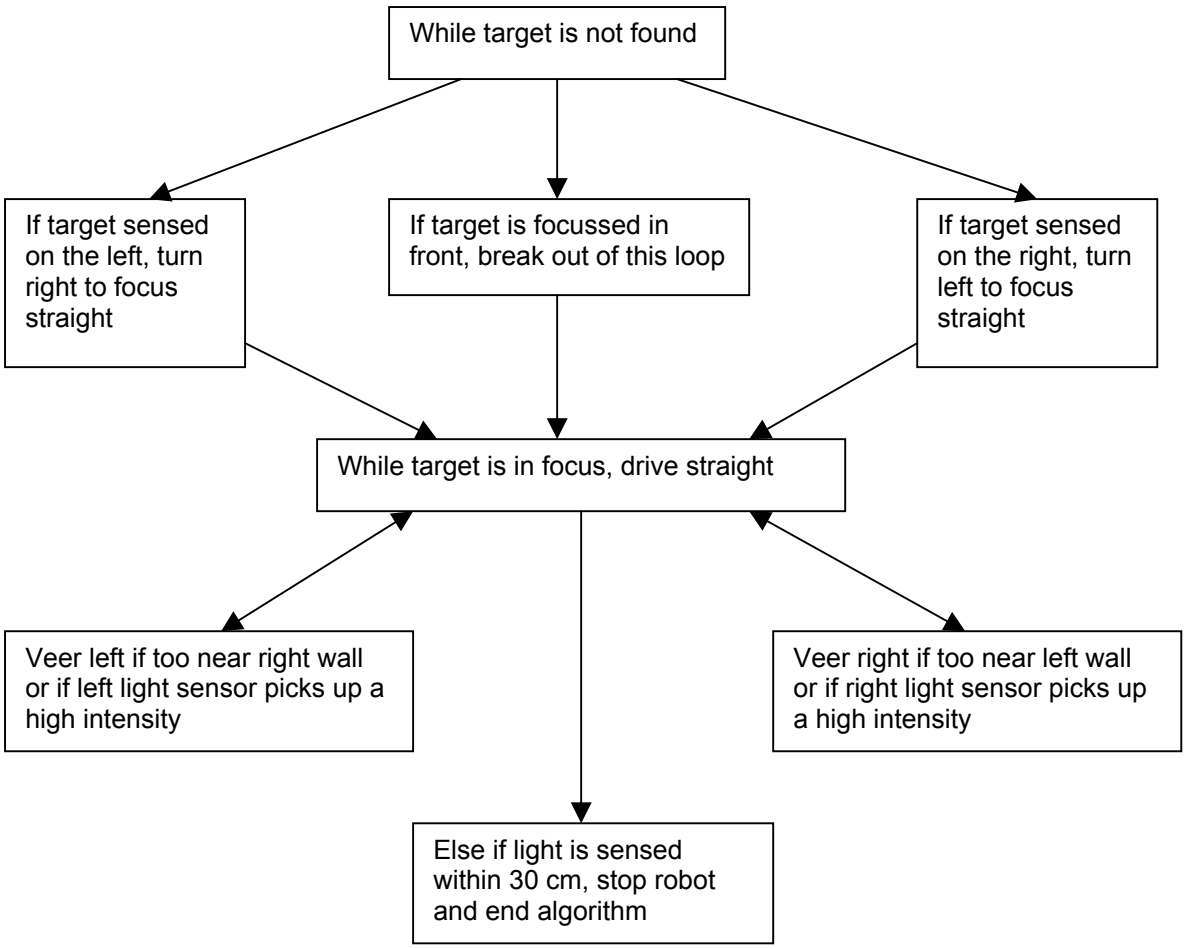


Figure 3.9: Search function

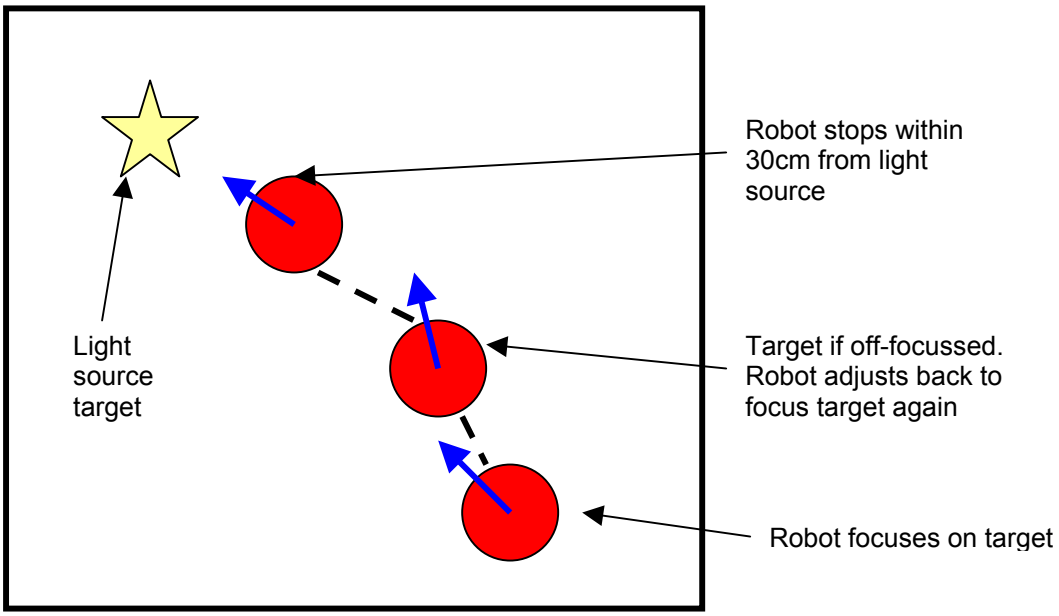


Figure 3.10: Manoeuvrability of robot when target if off-focussed

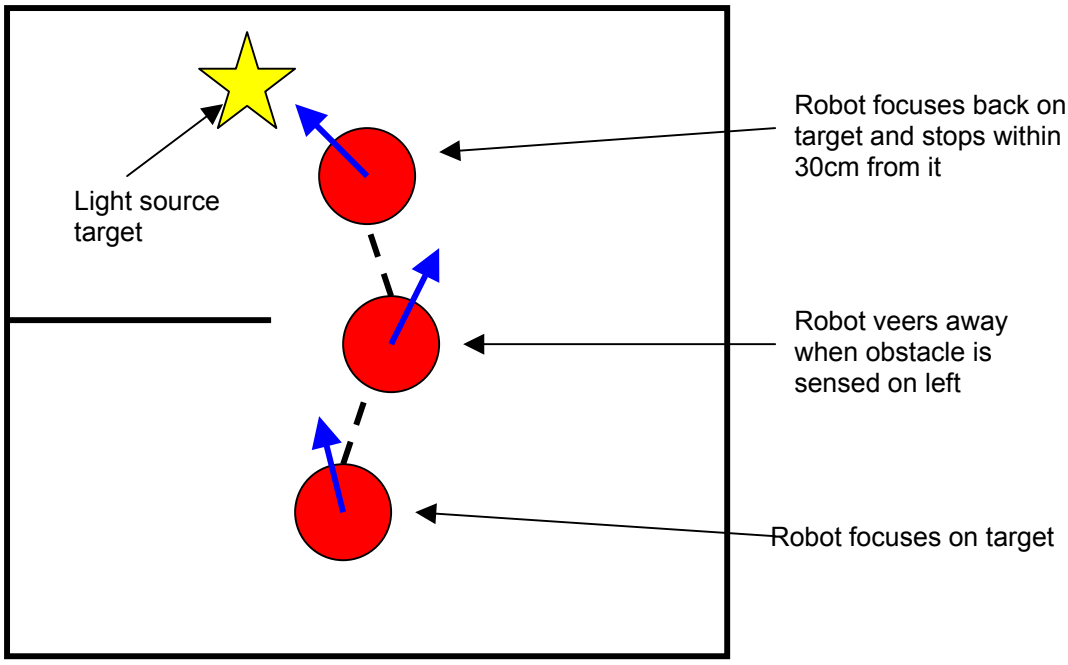


Figure 3.11: Manoeuvrability when obstacle is blocking robot's path to target

3.4 Search Algorithm 2: The 'most-open-area' search

The second algorithm runs on the concept of sparsity search. The robot will try to move into sparse areas when an obstacle blocks it from moving forward. The sparse area can either be in the "north", "north-east", "north-west", "east", "west", "south-east" or "south-west" directions. It should not retrace its own steps such that it moves back into the same explored area. Also, if while moving, the robot senses an "open" area on its right or left, it should stop and explore those areas as well. This allows the robot to cut across rooms and into empty spaces to explore. The iterative pseudo main code of this algorithm looks like this:

```
while (startflag)
{
        use the decision-making function (3.4.1) to decide which is "most-open"
        direction;
```

use the *move function* (3.4.2) which tells the robot to move;

}

3.4.1 Decision-making function

Inside the decision-making function, the robot decides which is the 'most-open' direction. This function makes use of an array sort of its 8 sonar values and returns the biggest value. The code then matches the biggest value to the direction it got the readings from. The "south" direction, which is the direction the robot initially arrived from, is not being considered, so that the robot does not retrace its steps. In order to cover larger spaces, the sorting array considers the more important directions last so that they overwrite and 'drop' into the largest array slot when exact same values of sonar readings are being returned. The "southeast" and "southwest" directions are considered to be more important at the point when the robot meets an obstacle and hence are being compared last in the sorting array. Once a direction has been picked, the robot turns to that specific angle needed to face in that direction. The exception is when the "open" direction returned is "north" and that most like implies that the sonar sensors have detected side obstacles. Since the side obstacles made the robot stop, its front is still relatively clear. To solve this problem, the algorithm has a "if" loop for the "north" direction and that means that if "north" is the "open" area, the robot is made to turn 30º right if the sensors detect nearer obstacles on the robot's left and turn 30º left if the sensors detect nearer obstacles on the robot's right. Figures 3.12 and 3.13 illustrate the concept of this function.

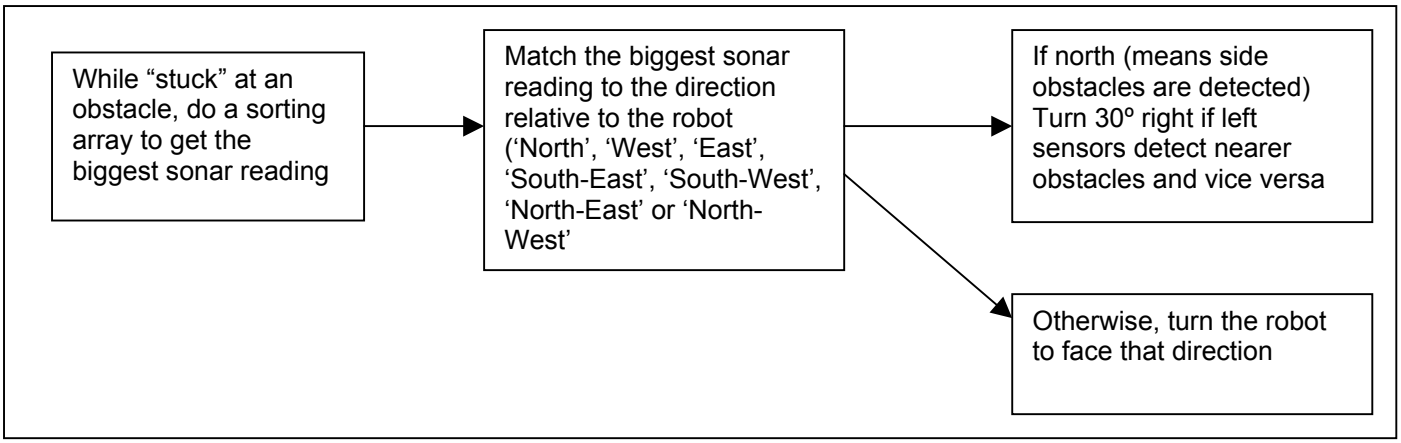


Figure 3.12: The decision function

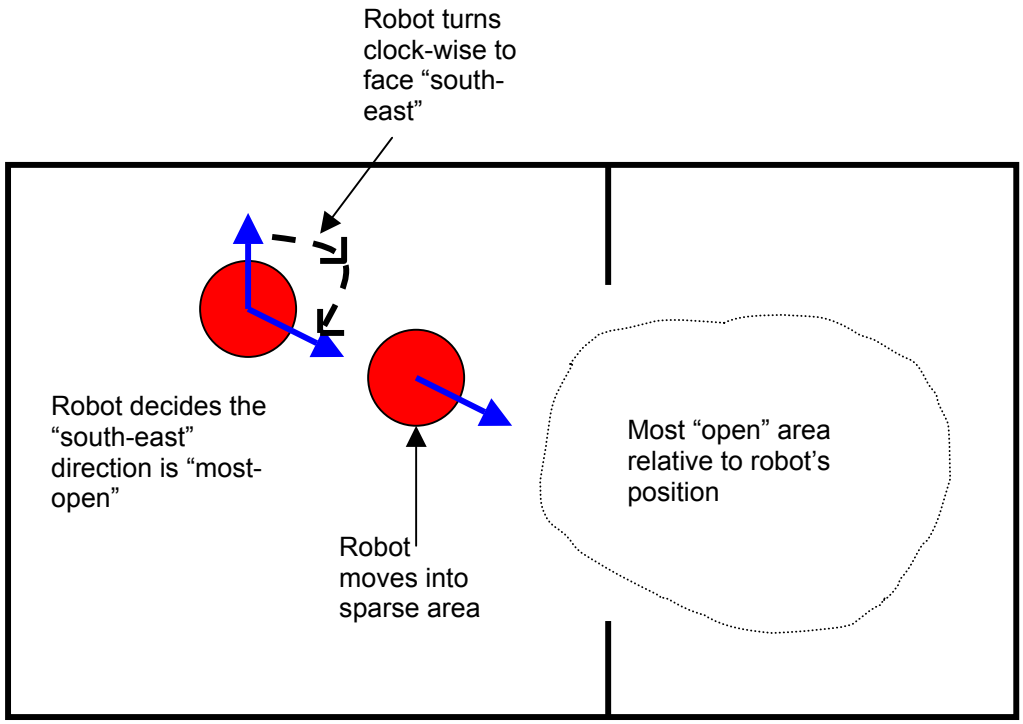


Figure 3.13: Picture illustration of the decision function

3.4.2 Move function

In this function explained in figures 3.14 and 3.15 below, the robot just drives straight while collecting sensors’ data along the way. An intermediate decision-making intelligence is fitted inside and that is, if the robot senses either its left or right (west or east) direction is “open”, it stops, turns and

moves towards that direction. This is important as it saves time and allows the robot to effectively cut across spaces and even rooms in its bid to find the target. Within this read-think-act loop, the robot also has an emergency obstacle avoidance method to avoid crashing into wall edges when it is too near them while moving through a door. If the robot senses a large intensity value, it goes into the search function, which is the same as that described in *section 3.3.4* and that allows the robot to locate and stop within 30cm from the light source.

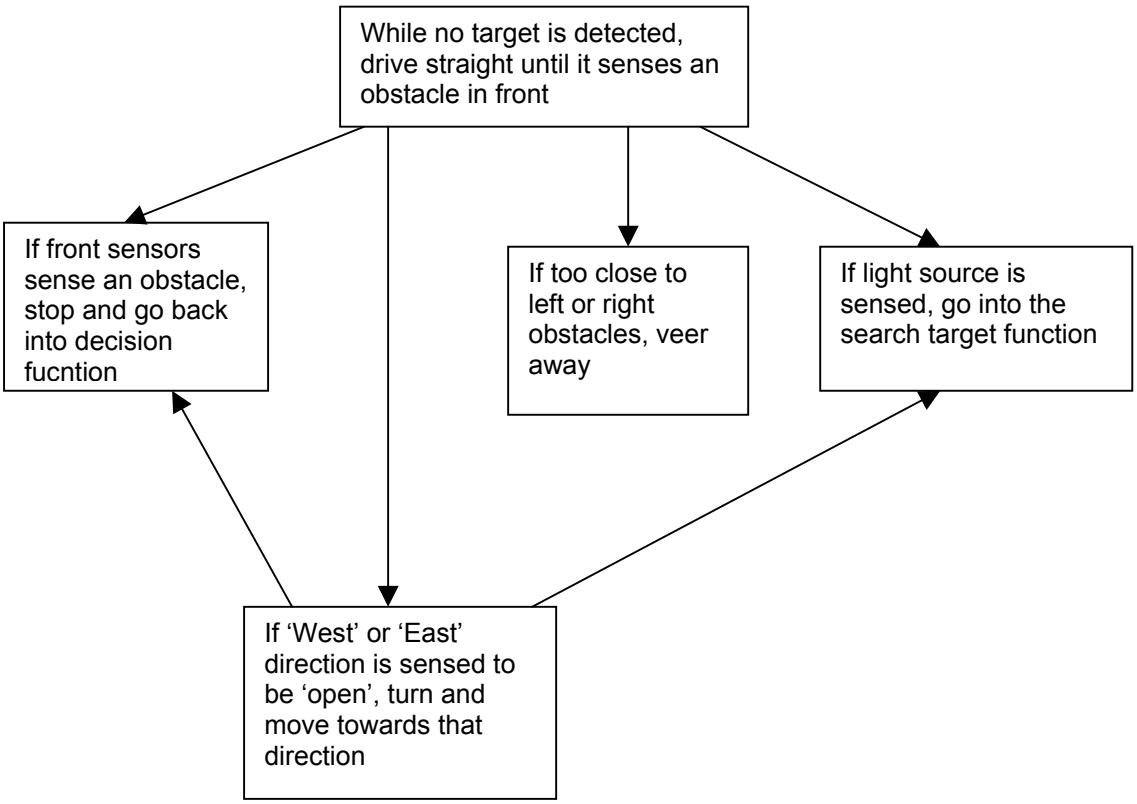


Figure 3.14: The move function

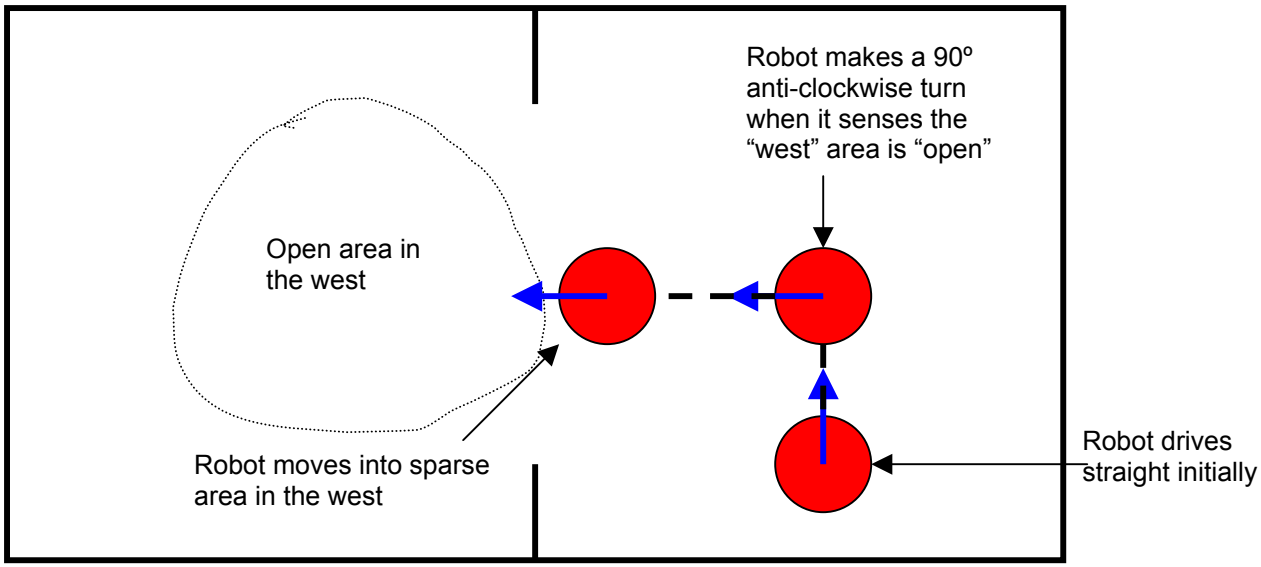


Figure 3.15: Illustration of the robot’s ability to explore the “west” or “east” directions

Chapter 4: Validation of algorithms through simulations

4.1 Stage settings

Player & Stage is used as the robot simulator and in this project, the Linux RedHat 9.0 Operating System provides the necessary interfaces needed for the simulations to run. Stage provides the necessary GUI tools for the creation of the environment and some physical entities' settings are shown in table 4.1

Table 4.1: Simulated entities' reference and GUI settings of the test-room

Room (A bitmap object)	Target (box object)	Robot (position device)
• *Size* - 10.625 m by 10.625m • *Colour* – black • *Sonar_return* – "visible" • *Laser_return* – "invisible • *Scale* – 0.021	• *Size* - 0.5m by 0.5 m • *Shape* – "circle" • *Colour* – black • *Sonar_return* – "visible" • *Laser_return* – "visible"	• *Size* - 0.5m by 0.5m • *Shape* – "Circle" • *Colour* – "Blue" • *Sonars* – 2-D, 24 rays, each with maximum range of 6m. • *Laser* – Maximum range of 4m, with maximum possible scanning radius of 180°. • *Drive* – Differential and non-holonomic

Figure 4.1 below shows a partial picture of the mobile robot and its environment. In order to test the algorithms, a test room of size about 10.625m by 10.625m is created. A relatively large round position device is used to simulate the mobile robot to allow for size compensation and adjustment in real practice. A box object, which "illuminates" when detected by the laser scanner, is used as the target. Since Stage does not provide light sensors, the laser device is used to simulate light sensors and its scanning range is set to 4m (in blue), corresponding to slightly less than half the width of the layout. Its scanning radius is extended to 180º, the maximum possible

limit. 24 two-dimensional sonar rays (in green) are used to simulate actual sonar sensors.

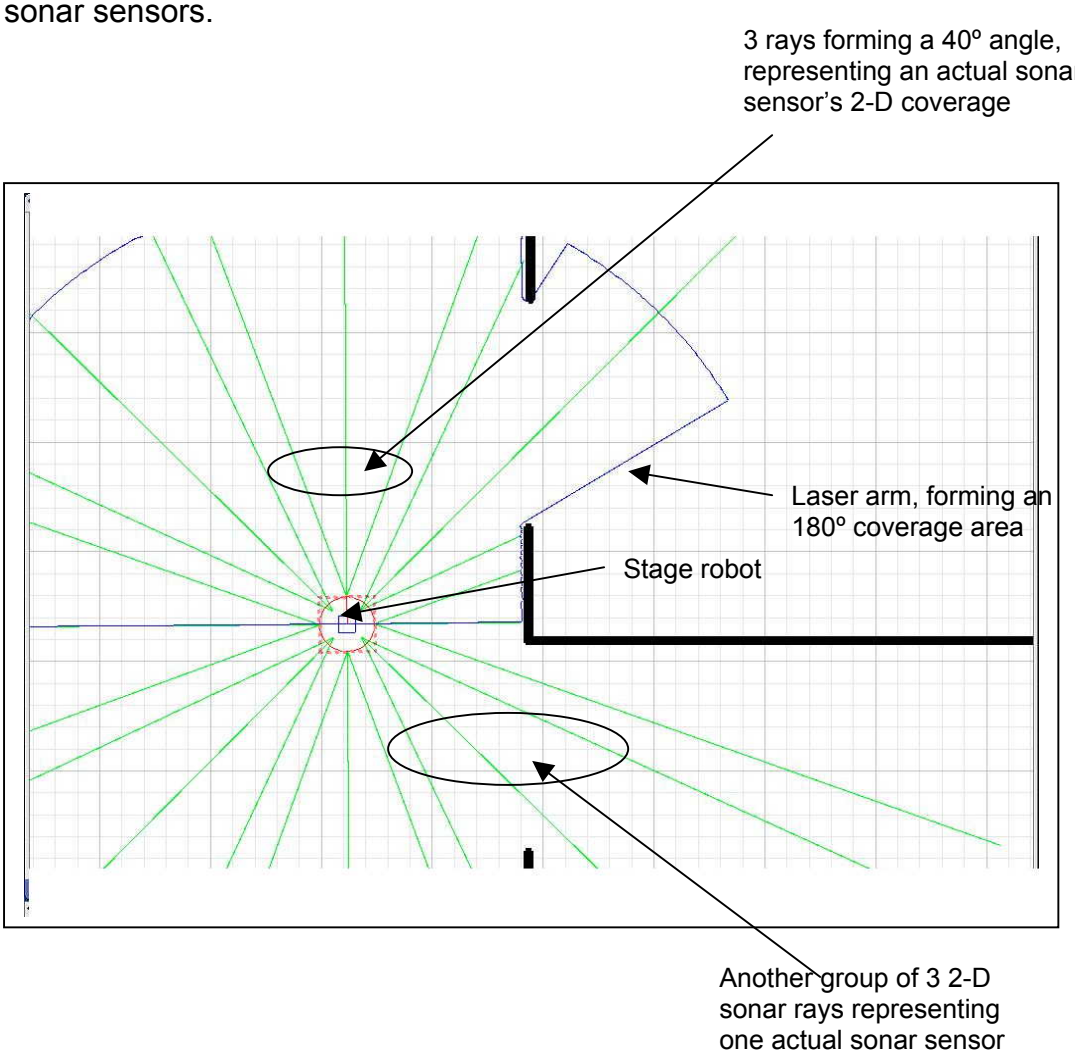


Figure 4.1: The stage robot and its environment

These sensors are grouped into 8 parts, with 3 rays forming each part and each part covering an angle of 40º. The 3 rays in each part spread out equally over 40º and the 8 parts are distributed equally over 360º round the robot. Thus, the discrepancy between the 2-D simulated sonar sensors and the actual ones, which have 3-D sensing capabilities, need to be taken note of. This is because the 2-D simulated sonar rays do not provide a volumetric coverage of its span, resulting in large non-covered areas (in white) between all the single rays.

The size of room openings is made flexible, at about 4 to 5 times that of the size of the robot. This is to simulate the real robot against an actual door width. There are two ways to make use of the laser device to detect the target. One is to create an object, which is “visible” to the laser beam and returns a positive intensity value (that is ‘1’) when the laser beam falls on it. The other is to impose a fiducial identification number (id) on the object and when the laser beam falls on the object and it can return the id number to the program. Both methods require the use of the laser device, and therefore they share many similarities. Since the fiducial finder device is a child of the laser device, the first method is chosen as it lessens the complications. Again, it is imperative to note that this method does not fully simulate what happens in the hardware tests. For instance, the robot in Stage will respond immediately to the slightest of detection of its the target with the slightest of touch of it laser-scanning beam against the target. Also, the intensity value in Stage only has two values (‘1’ and ‘0’), which is unlike the situation in reality where the intensity of the light source has a wide range of values.

It is possible to set the cycle update period (in seconds) with reference to the wall-clock time (real-time) in Stage. If the cycle’s computation takes longer than the requested cycle time, Stage will run slower than requested. Also, Stage can also be made to simulate the passing of simulation time-step (in seconds) per update cycle (simulated-time). Thus, by adjusting the ratio of the real and simulated time, Stage can be made to run approximately at real-time. In this project, both simulated and real timings have update periods of 0.1 seconds. But, there are times when the cycle update period has to be

increased to allow for Stage's long computational cycle time. If this is not taken note of, Stage may not be able to communicate with Player fast enough to update its sensors. This is analogous to a person running too fast, and not having enough time to think and hence crashing into a wall, which he would have avoided if he had been slower.

4.2 Player Client program

The Player Client program is written in the C++ language and it interacts with the server (Stage robot) in a client-server relationship. Information is exchanged via the Player Client proxy (please see appendix 9.1 for more information). The main code must begin with the initialisation of relevant proxies used in the client program. An example of this initialisation is shown in the partial code in figure 4.2. These proxies will enable the simulated robot to have the necessary abilities to perform the algorithms.

```
int main(int argc, char **argv)
 {
   PlayerClient robot("localhost");
   SonarProxy sp(&robot,0,'r');
   LaserProxy lp(&robot,0,'a');
   PositionProxy pp(&robot,0,'a');

   /*Variables used in LaserProxy*/
   bool intensity;
   short min_angle, max_angle;
   unsigned short resolution;

   /*Distance variables*/
   unsigned short min_front_dist = 400;
   unsigned short parallel_dist = 600;

       /* go into read-think-act loop */
   while (startflag)
   {
    /* this blocks until new data comes; 10Hz by default */
    if (robot.Read())
    exit(1);
/* Codes continues here…..*/
```

Initialising of proxies

Figure 4.2: Example of a partial code to start the program

4.3 Simulated Algorithm 1 (Wall-following search)

Figures 4.3(i to vii) below show the chronological simulated paths, depicting how the robot moves and searches for the target. The wall-following speed is set low at 0.1m/s to give itself some time to adjust when it is too close or too far from the wall. Equipped with sonar (shown in green) and laser (blue) sensors, the robot follows the wall in figures 4.3(i) and 4.3(ii). In Figures 4.3(iii) and 4.3(iv) and (v), the robot makes a 90º concave corner turn and a 180º convex corner turn respectively. In figure 4.3(vi), the robot spots the target and narrows its laser-scanning radius to 40º in order to pin point the target. Once focussed on the target, the robot moves towards it and stops within a certain distance of about 0.3 from it. Likewise, figures 4.4(i to vi) and 4.5(i to iv) show the wall-following algorithm using different starting points and different target positions.

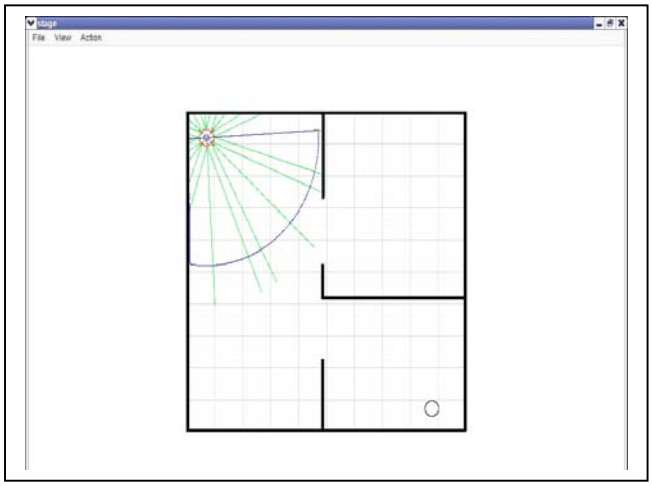


Figure 4.3(i)

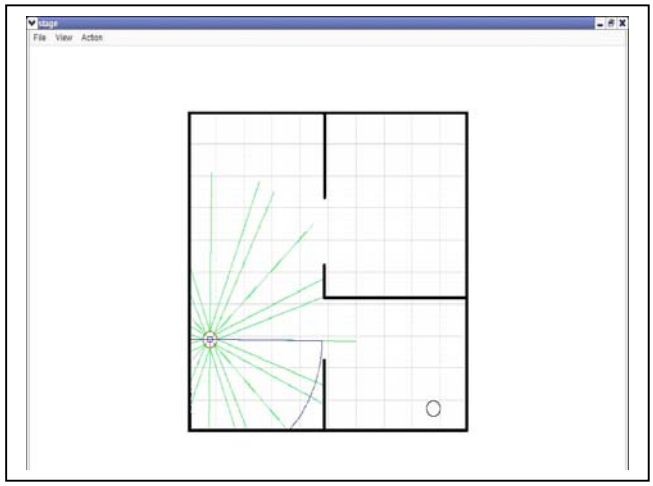


Figure 4.3(ii)

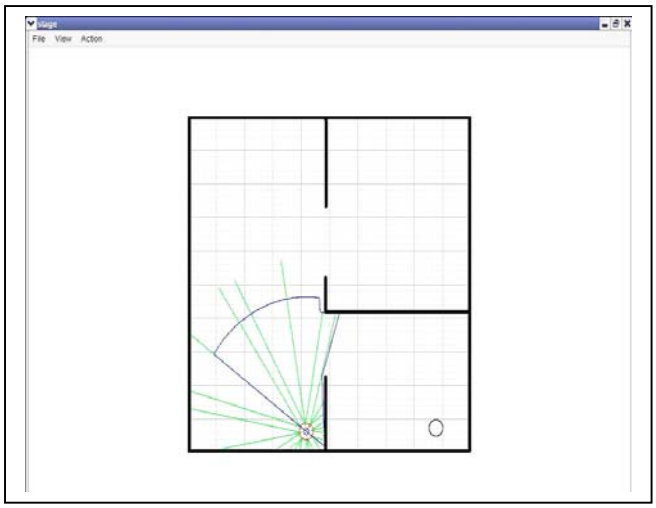


Figure 4.3(iii)

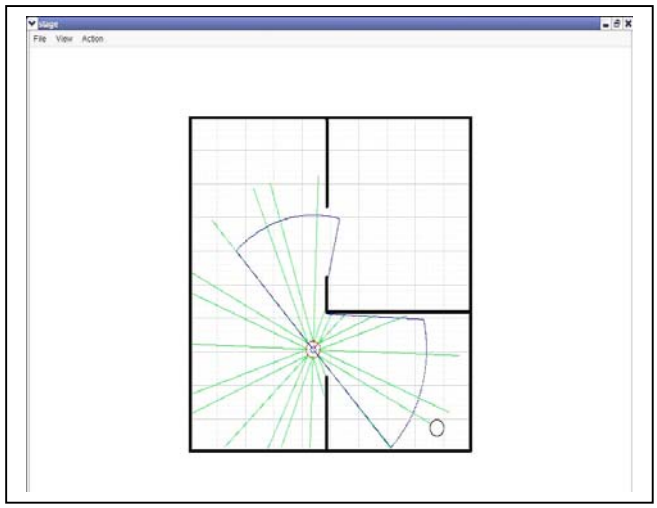


Figure 4.3(iv)

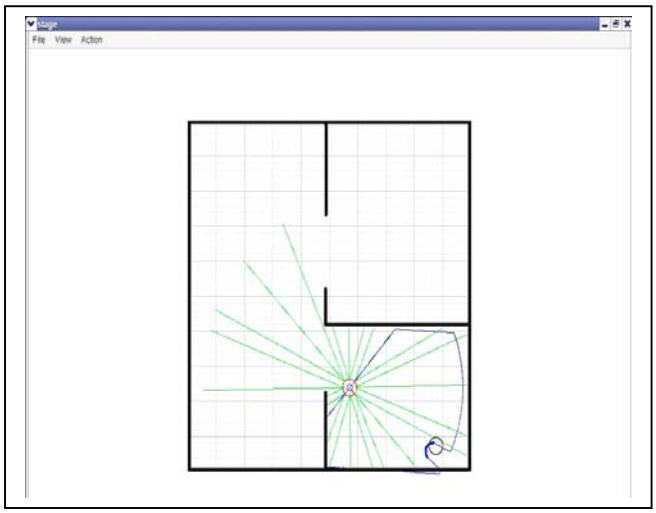


Figure 4.3(v)

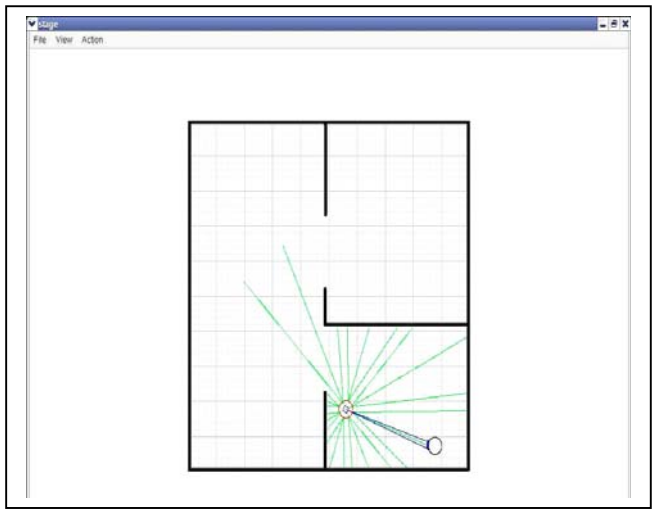


Figure 4.3(vi)

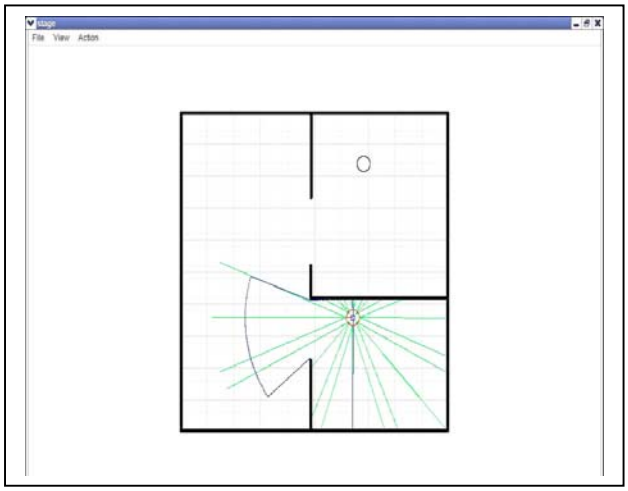


Figure 4.4(i)

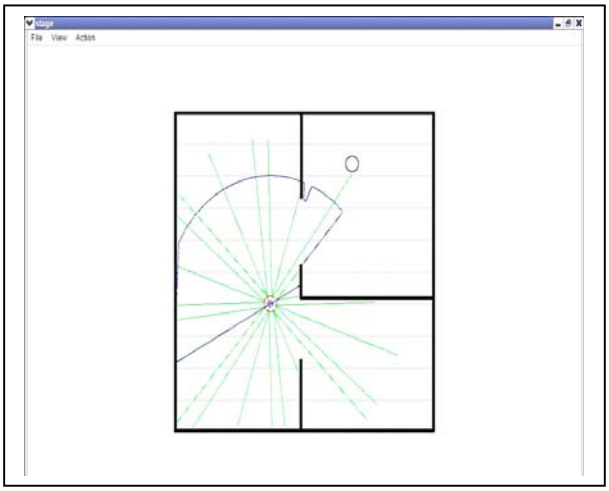


Figure 4.4(ii)

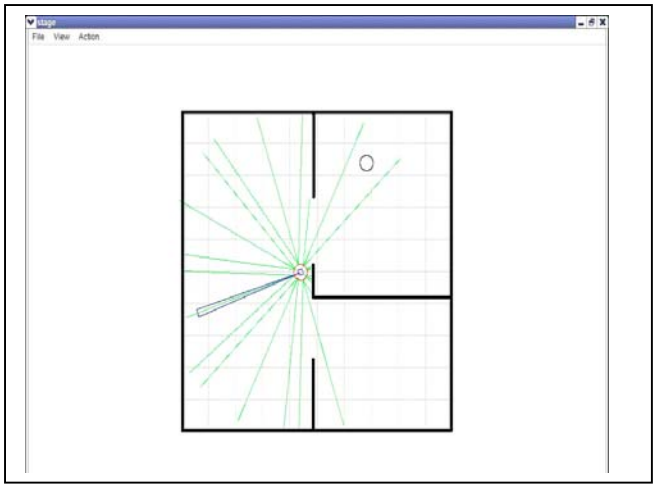


Figure 4.4(iii)

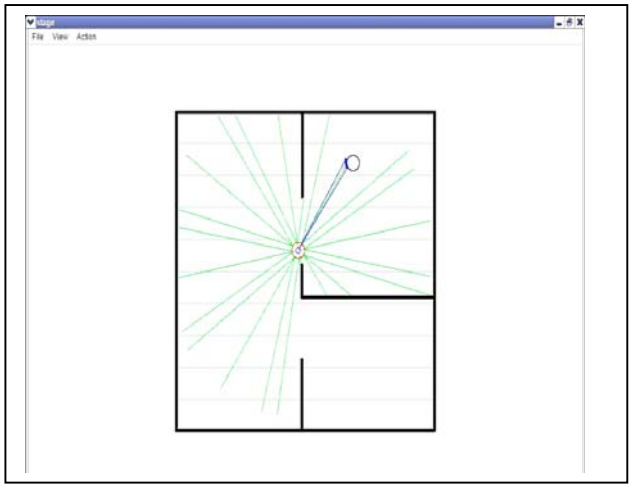


Figure 4.4(iv)

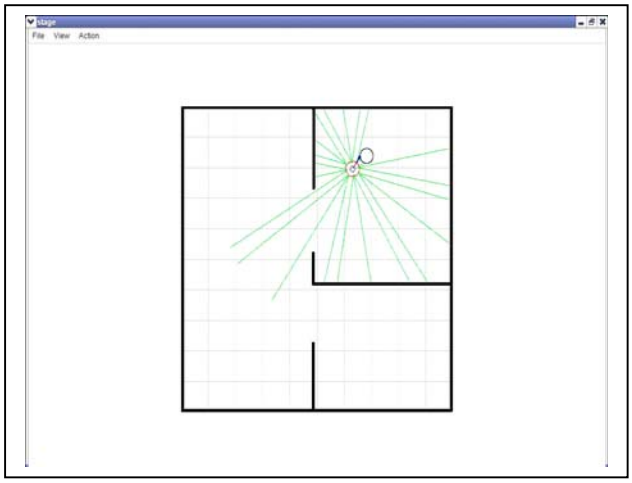


Figure 4.4(v)

The robot is also programmed with obstacle avoidance ability as it moves towards the target so that it does not crash into the wall as it moves towards the target.

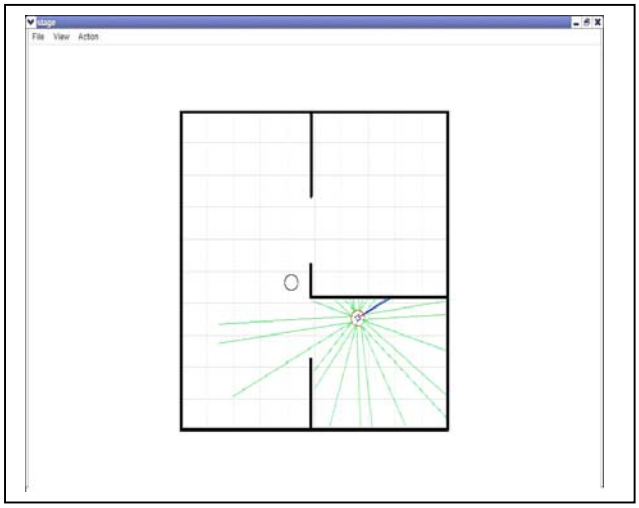


Figure 4.5(i)

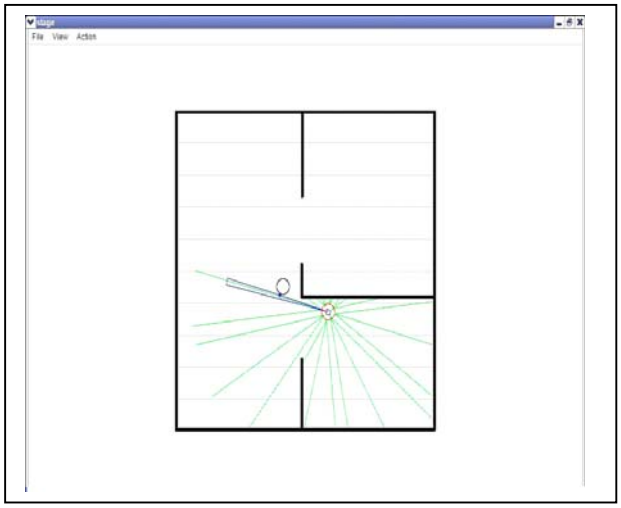


Figure 4.5(ii)

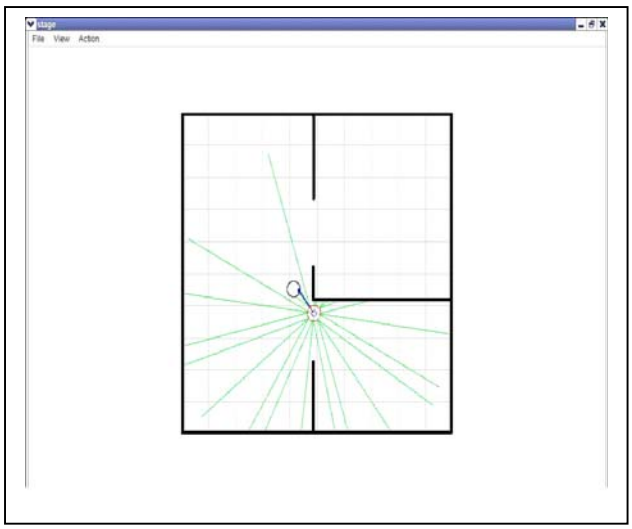


Figure 4.5(iii)

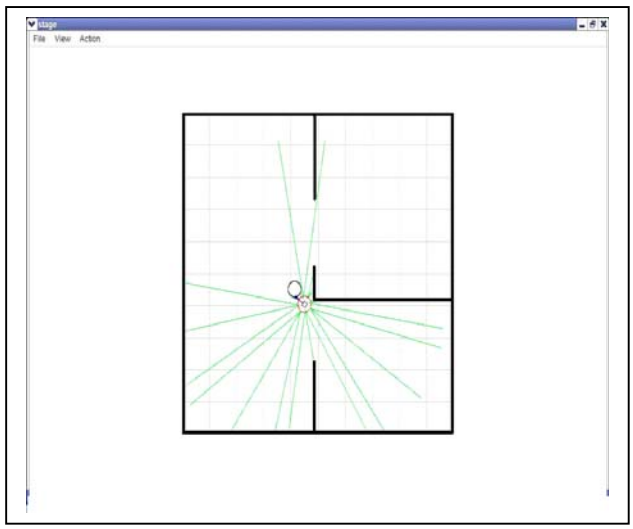


Figure 4.5(iv)

4.4 Simulated Algorithm 2 (The “most-open-area” search)

Similar to that of section 4.3, these diagrams in this section illustrate how this algorithm works in a chronological order, with different starting points and target positions. It should be noted that the value processed and considered in the *decision function* for a particular direction is the sum of the values of 3 single sonar rays, which have been grouped together to act as an actual sonar sensor.

In figures 4.6(i) and 4.6(ii), the robot is able to detect the door opening as it senses that direction is “most-open” and thus turns and moves towards that direction. A slight faster (compared to wall-following speed) moving speed of 0.3m/s is set for the robot, as it does not have the wall as a compulsory guide. The robot performs the same logic thinking in figures 4.6(iii) and 4.6(iv).

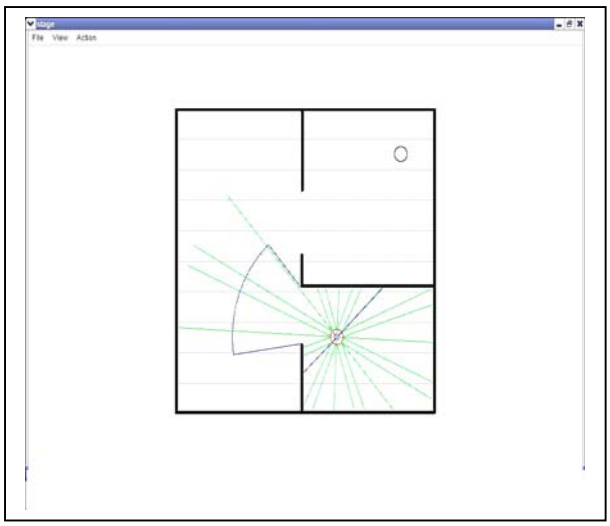


Figure 4.6(i)

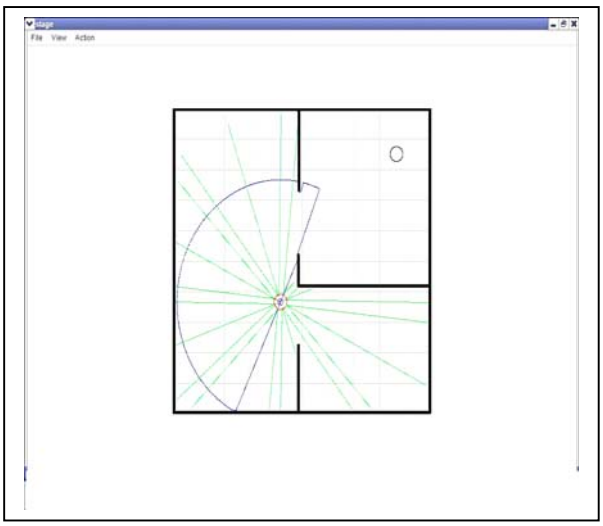


Figure 4.6(ii)

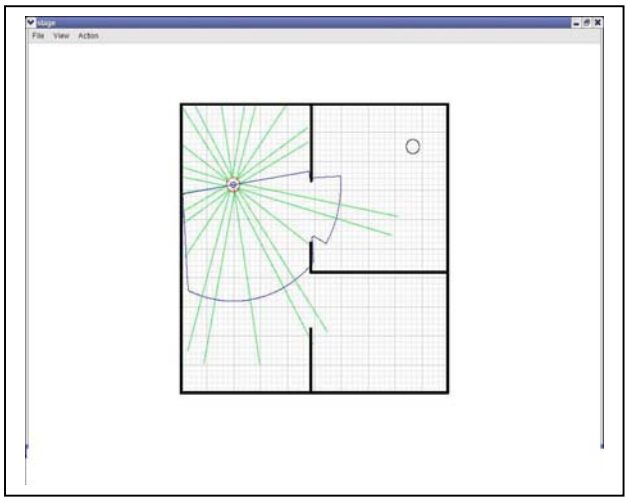


Figure 4.6(iii)

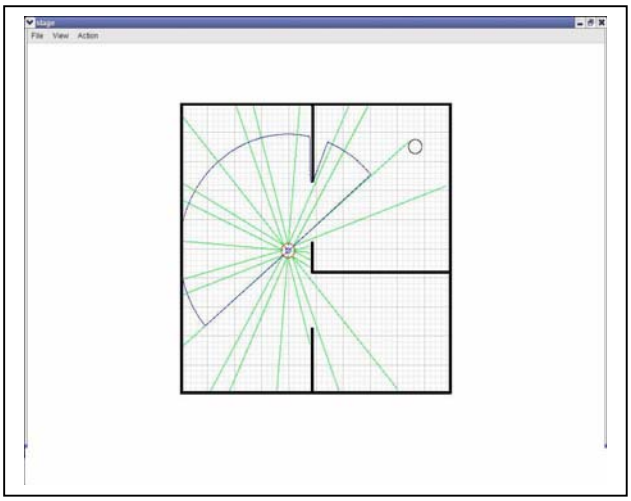


Figure 4.6(iv)

In figures 4.6(v) and 4.6(vi), the robot manages to move into the room which contains the target, narrows its laser scan on and drives towards it.

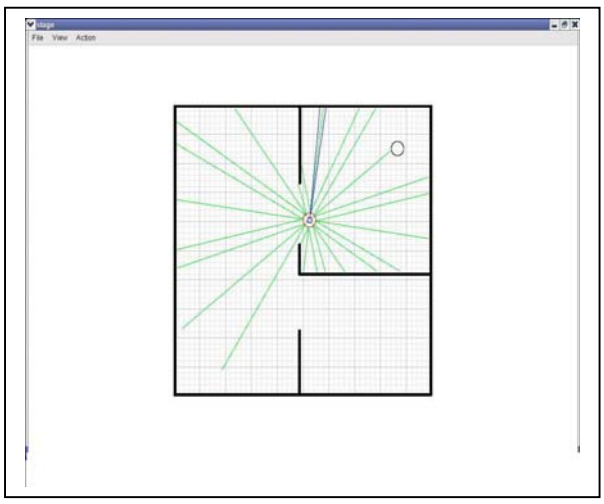


Figure 4.6(v)

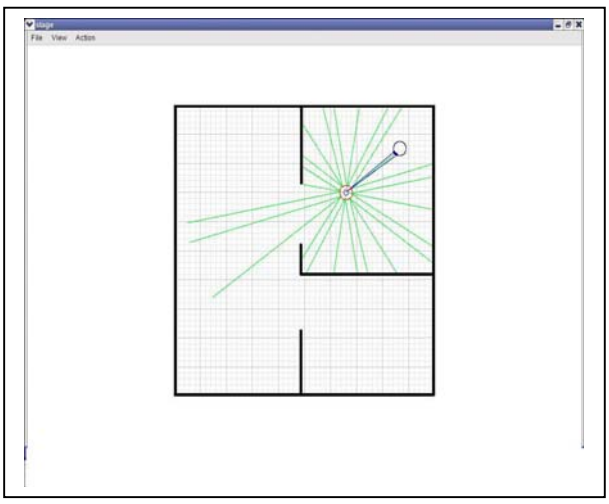


Figure 4.6(vi)

Figures 4.7(i to v) show another simulated run-up of the “most-open-area” search algorithm method with a different robot starting point and a different target position.

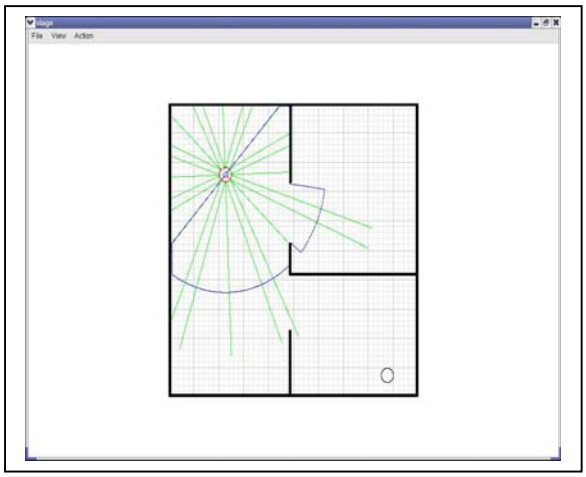


Figure 4.7(i)

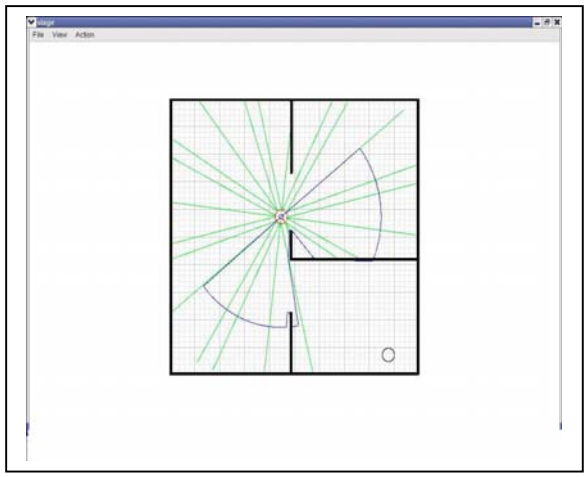


Figure 4.7(ii)

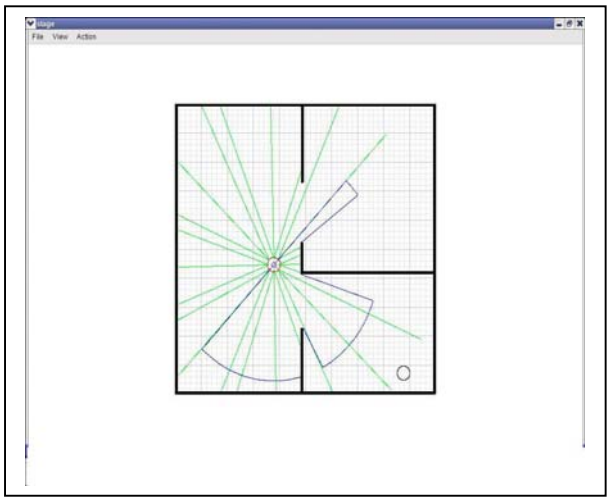


Figure 4.7(iii)

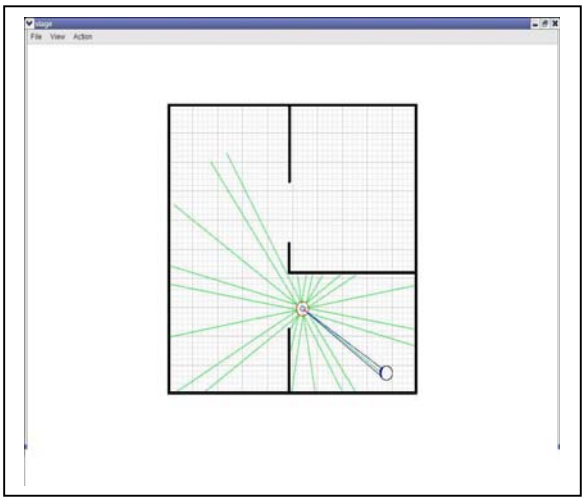


Figure 4.7(iv)

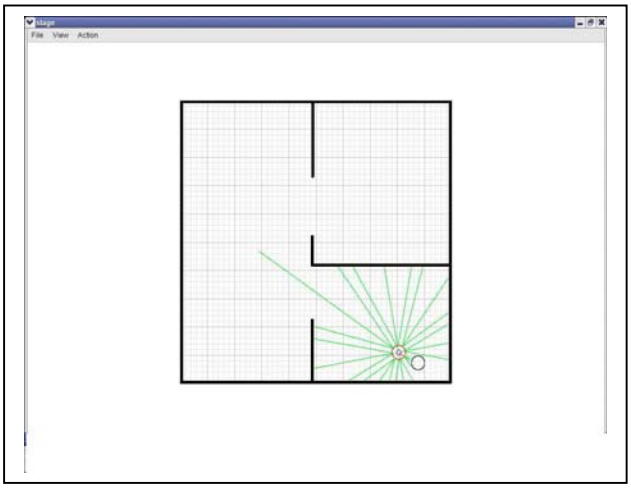
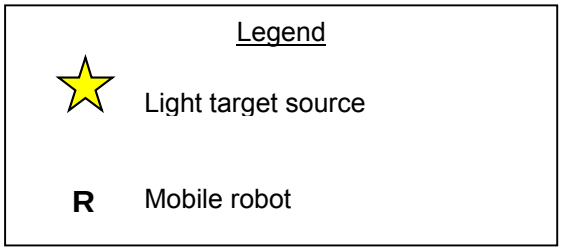


Figure 4.7(v)

4.5 Experimental methods and results of simulations

The experiments are conducted using 14 different combinations of robot starting and target positions shown in figures 4.8(i) to 4.8(v). A wall-clock imbedded in the program (similar to a stop-watch) is used as the timing device to obtain the time taken for the robot to locate the light source target. This method is justifiable because the simulation clock is set to be as close as possible to the real-time clock (at 0.1s update period). Each combination is used to test both algorithms 5 times and the average time taken is recorded. These average values can then be used in the analysis of the search methods. An experiment "failure" is said to occur when the robot takes too long to locate the light source and in these simulations, an unsuccessful test occurs after 12 minutes of futile search. Therefore success rates (out of 5 tests) of the algorithms are also noted. The detailed results are displayed in appendix 9.3.



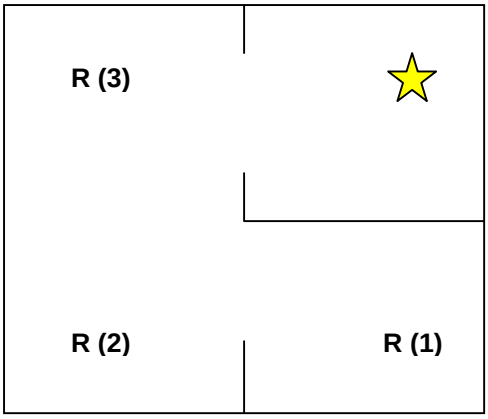


Figure 4.8(i): Target in room with 3 different robot starting positions

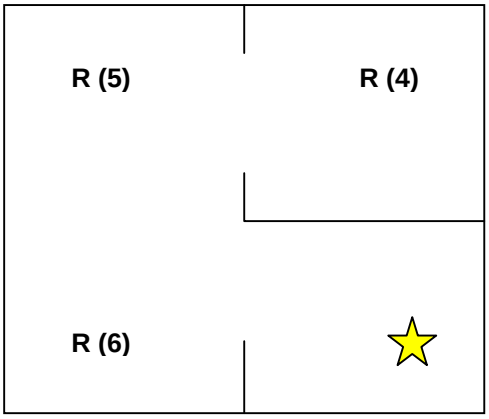


Figure 4.8(ii): Target in another room with 3 different robot starting positions

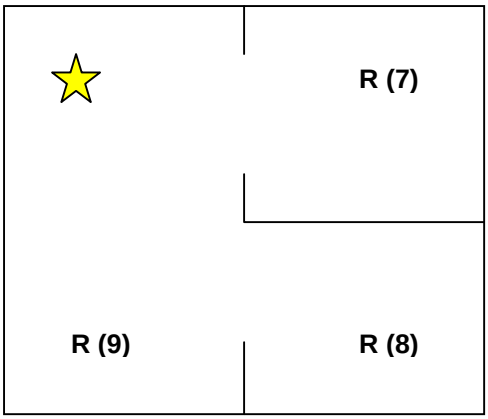


Figure 4.8(iii): Target in main area with 3 different robot starting positions

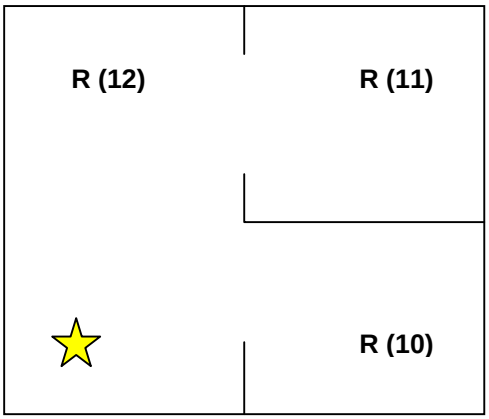


Figure 4.8(iv): Target in another corner of main area with 3 different robot starting positions

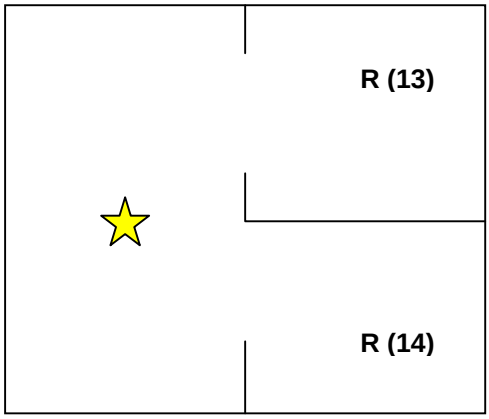


Figure 4.8(v): Target in centre of main area with 2 different robot starting positions

4.5 Analysis of simulation results

From the experimental results, we can see that the “most-open-area” search algorithm recorded faster times (faster times are high-lighted in table 4.2) in the majority of the experimental combinations. However, it should be noted that the success rate of this algorithm is not always 100% (that is it either takes longer than 12 minutes to complete the search or never at all). The wall-following search method though slow most of the time, is methodological and steady in this sort of environment. Its 100% success rate grinds in the point that it is extremely efficient and if the search time is not an important

consideration factor. Tests done for combinations 5,6 and 12 even showed that the wall-following search method is able to provide faster search times in a few cases.

The “most-open-area” search algorithm presents its case as a more random and risky search method, depending on the robot’s ability to detect room openings. Still it provides faster search times to the former search method in most scenarios done in this experiment. Therefore if risks can be taken in extremely urgent cases, this method could be useful for such uses.

Table 4.2: Tabulated results of experiments in simulations

Combination No.	Average time-taken for search		Success Rate (out of 5 tests)	
	Wall-following	“Most-open-area”	Wall-following	“Most-open-area”
1	3 mins 19 secs	3 mins 7 secs	100%	80%
2	6 mins 32 secs	2 mins 43 secs	100%	60%
3	8 mins 46 secs	3 mins 58 secs	100%	40%
4	7 mins 53 secs	3 mins 16 secs	100%	100%
5	5 mins 10 secs	7 mins 12 secs	100%	60%
6	3 mins 20 secs	4 mins 1 sec	100%	60%
7	2 mins 17 secs	1 min 32 secs	100%	100%
8	4 mins 49 secs	1 min 46 secs	100%	100%
9	7 mins 40 secs	3 mins 29 secs	100%	100%
10	1 min 39 secs	47 secs	100%	100%
11	3 mins 59 secs	2 mins 9 secs	100%	100%
12	1 min 11 secs	3 mins 4 secs	100%	100%
13	2 mins 27 secs	1 min 8 secs	100%	100%
14	2 mins 51 secs	1 min 36 secs	100%	100%

Chapter 5: Implementation of algorithms using hardware

5.1 Designing the mobile robot

A team of 4 final-year students, including myself were involved in the design of a suitable robot platform for use in our projects. It was decided that the mobile robot's system architecture would make use of the Brainstem technology developed by Acroname Inc (please refer to appendix 9.4 for more information). The brainstem GP 1.0 chip used in this project is provided by Acroname Inc. for the production of the PPRK (Palm Pilot Robot Kit) shown in figure 5.1 for the Carnegie Mellon University.

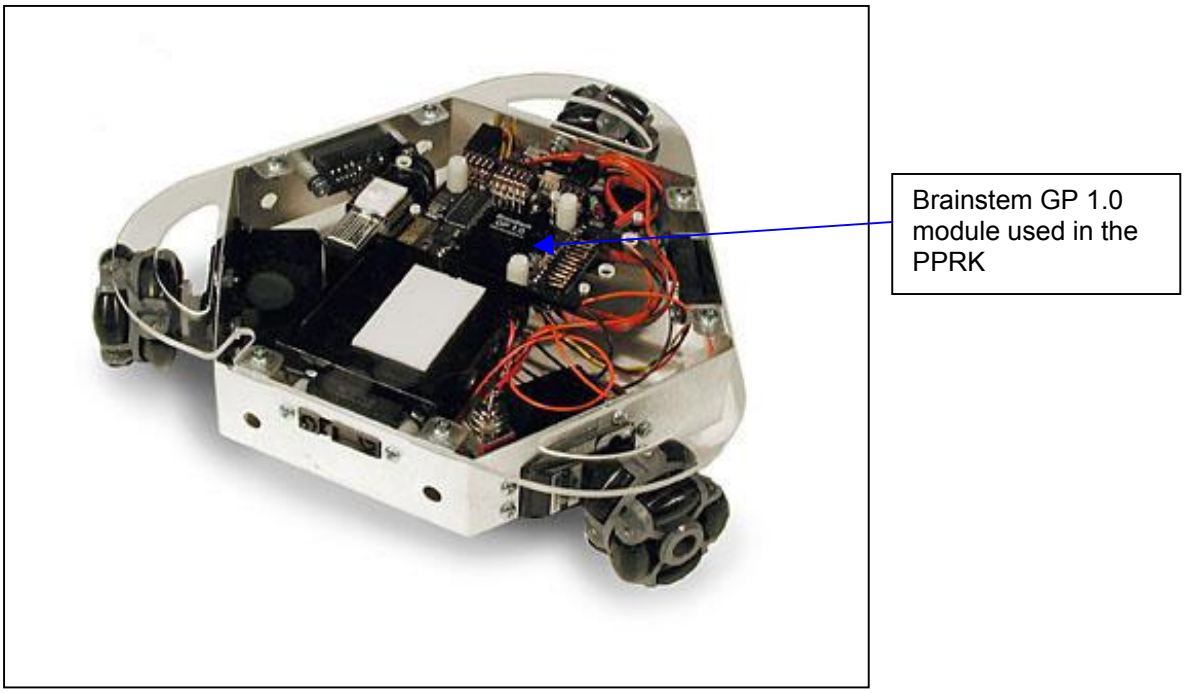


Figure 5.1: The Palm Pilot Robot Kit

The team decided to tap on the PPRK technology and the Brainstem GP 1.0 module to construct such a robot platform because of the availability of PPRKs and Pocket PCs in the Co-operative Systems laboratory.

However, modifications had to be done to the existing robot in figure 5.1 because such a robot does not provide the necessary platform to house the

necessary sensors. A properly suited robot must provide a platform for the sensors to work properly, without compromising on the driving mechanism. Thus, several initial robot designs were brainstormed and the first few of which turned out to be unsuitable platforms because of mainly manoeuvrability, stability and sensing problems. The first workable robot prototype (please see figure 5.2) to be tested was one, which was able to perform simple functions like making turns and moving straight without giving too much problems.

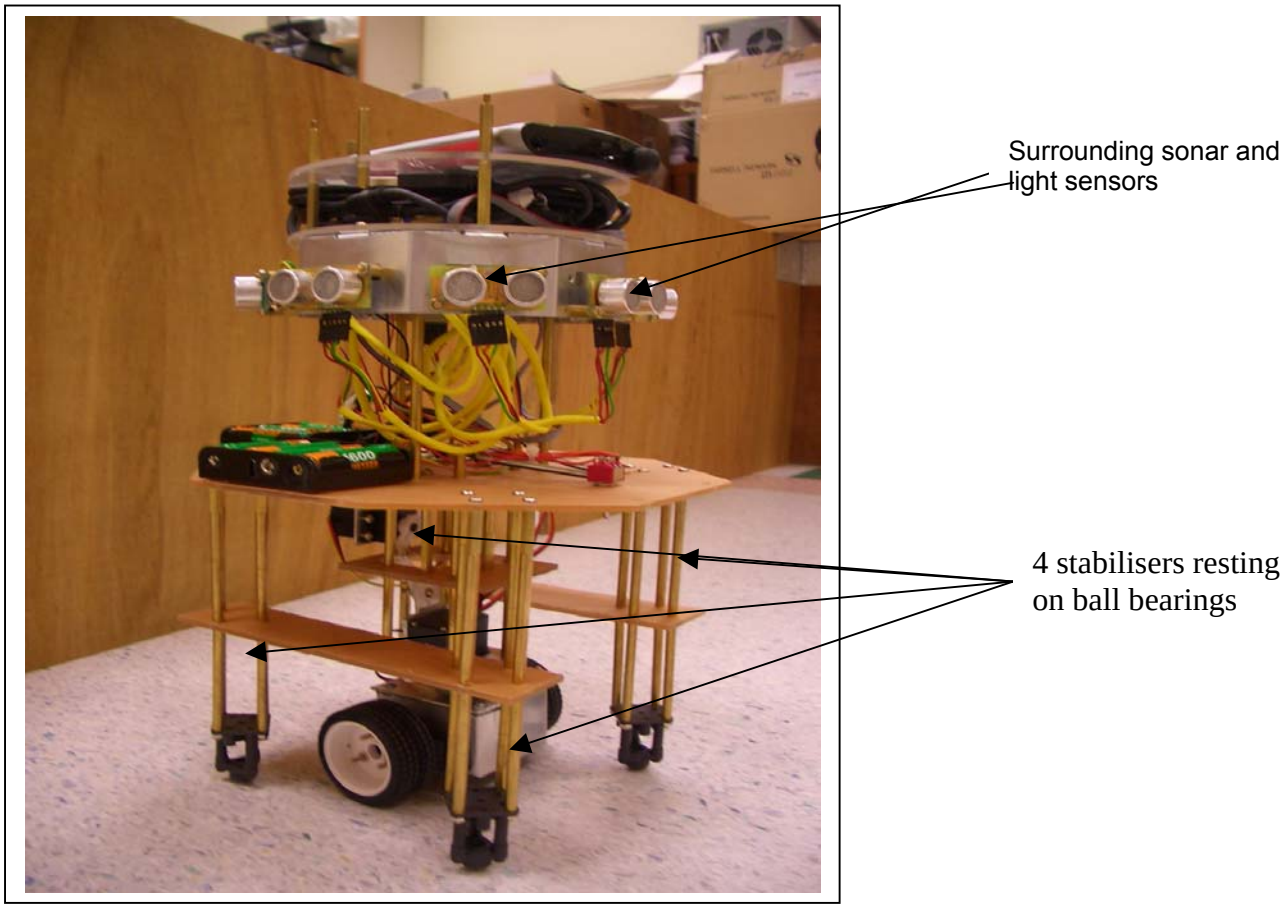


Figure 5.2: First tested robot prototype

However, the 23cm wide and 36cm high robot was just a little too bulky to make smooth turns and its 4 supporting ball-bearing stabilisers caused too much unnecessary friction, which led to jerks while turning. Eventually, after

preliminary tests, this particular robot design was decided not suitable for this project.

What was eventually used as the hardware robot is a finished design by Chee Kong, a postgraduate student. It is very similar to the first prototype, in the sense that it has 16 surrounding sonar sensors, including 8 surrounding photoelectric light sensors and a magnetic compass. The major differences are that it has only 2 supporting ball-bearing stands, which provides better manoeuvrability because of reduced friction and it is also smaller, with a height of about 29cm and a width of about 15cm and that makes it less clumsy as a mobile robot (please see figure 5.3 below).

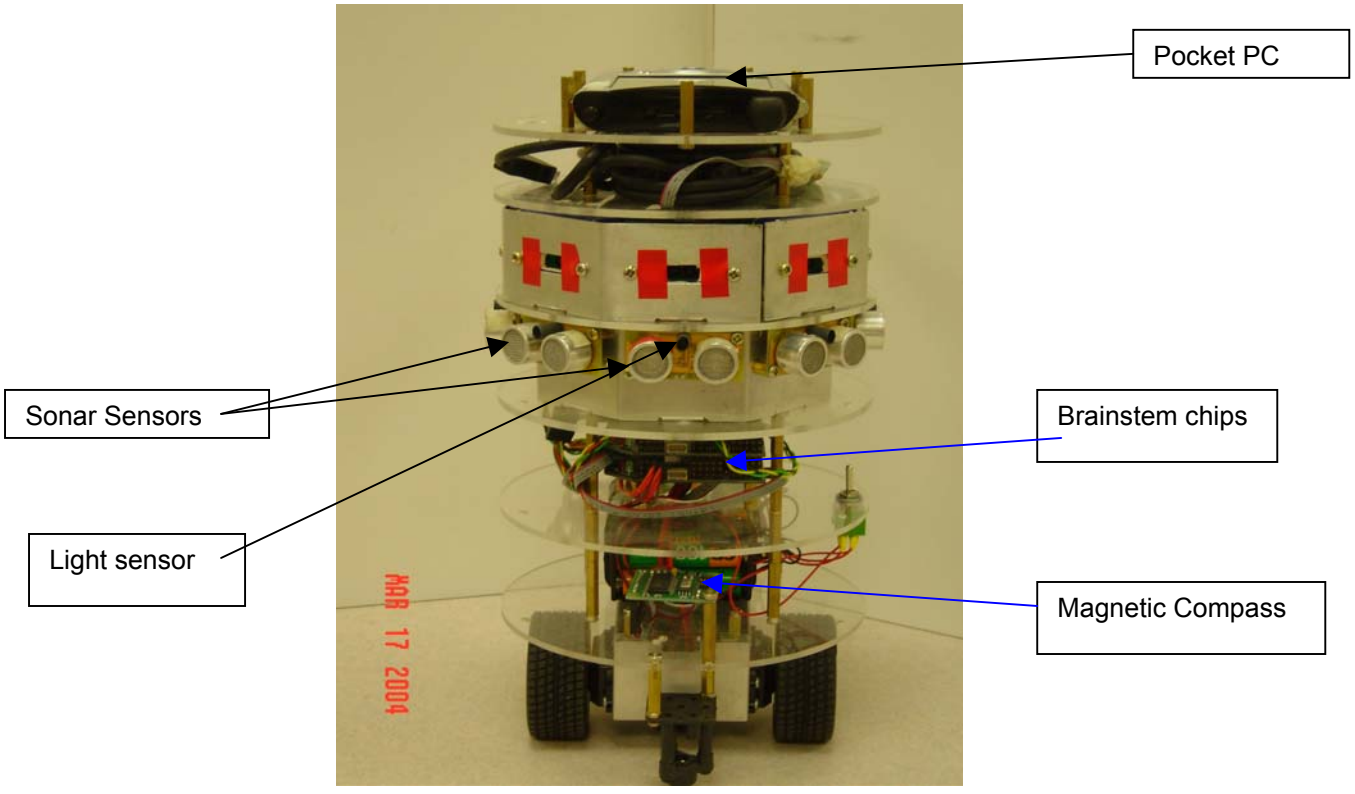


Figure 5.3: Robot prototype used in hardware experiments

5.2 Introduction to the components of the mobile robot

A decentralised command approach is adopted whereby a mounted pocket PC (please see figure 5.4 below) on top of the robot serves as the host and double up as the robot's brain. It interacts with the brainstem module in a host-client relationship in controlling the servos and the mounted sonar and photoelectric sensors.

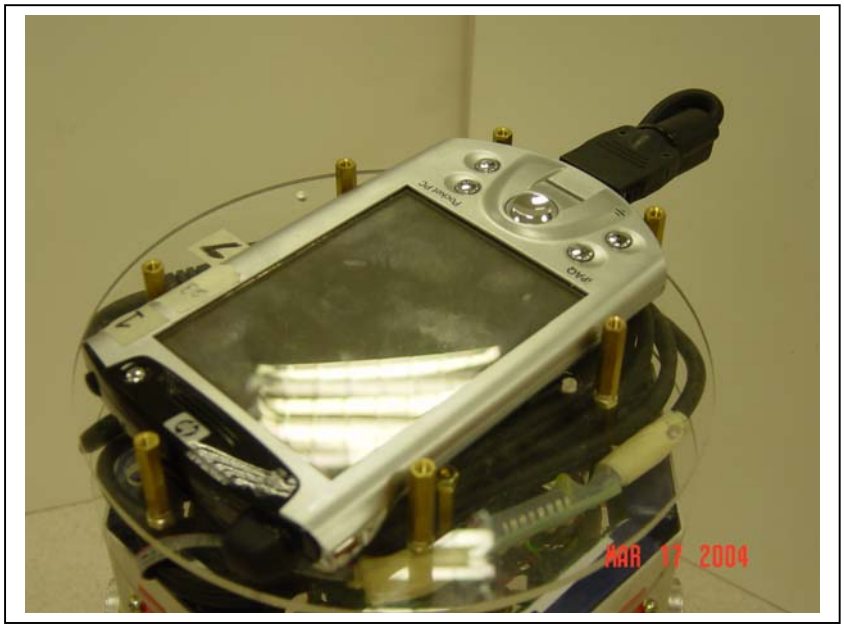


Figure 5.4: Mounted Hewlett Packard Pocket PC

The sensor used is the Acroname Devantec SRF08 RangeFinder as shown in figure 5.5, which has a combination of two fixed-on sonar sensors and as well as a photoelectric light sensor in between them. This model is particularly useful for this project because it incorporates both sonar and photoelectric sensors, thus serving the project's requirements purposefully.

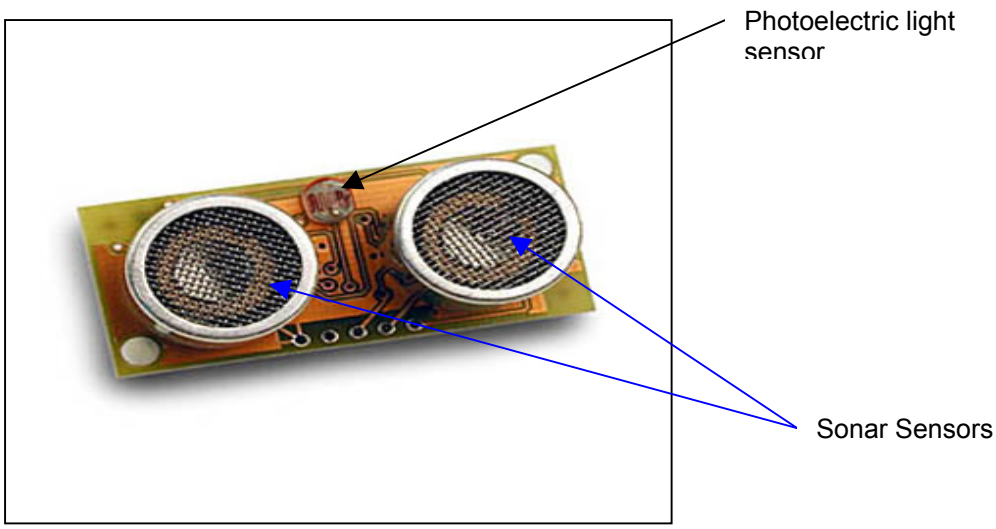


Figure 5.5: The Devantec SRF08 RangeFinder

Each sonar sensor has an effective range beam covering roughly 80º from left to right as shown in figure 5.6 below and the sensing range goes from 3cm up to 6m. However, pre-experimental tests showed that the sensing range sonar sensors are only accurate between 5cm and110cm.The robot also has an attached Devantec magnetic compass, which it uses for directional sensing and turning. (Please refer to Appendix 9.5 for more information regarding the sensors)

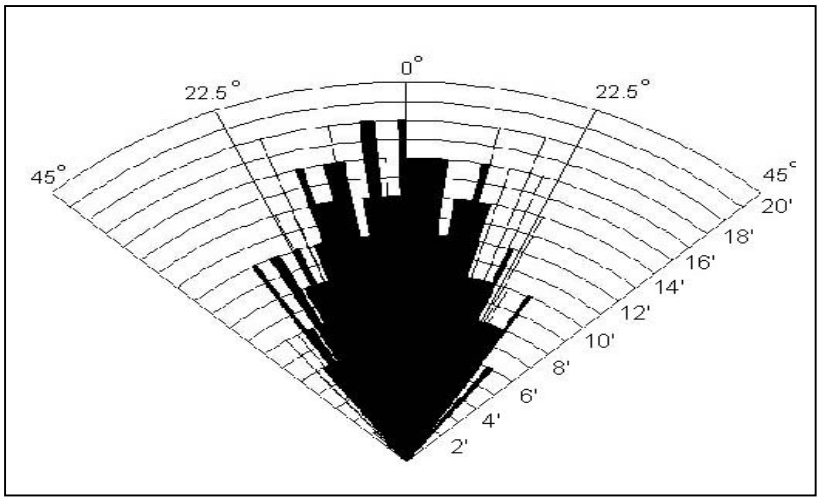


Figure 5.6: Sonar beam pattern

The compass used is the Philips KMZ51 magnetic field sensor, which has enough sensitivity to detect the Earth's magnetic field. This compass can be seen in figure 5.7.

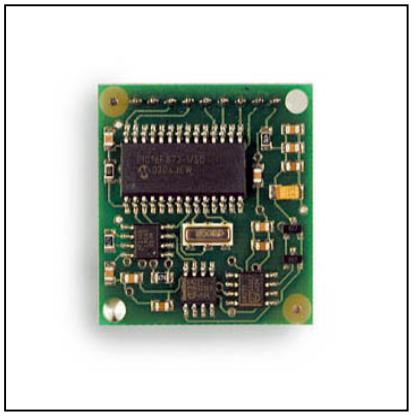


Figure 5.7: Devantec compact magnetic compass

5.3 Experimental layout mock-up

The hardware tests are conducted in the Co-operative Systems laboratory. Walls are represented by wooden planks, 0.4m in height and they are arranged to form a flat or office like environment for the robot to perform the search. The corners are made up of 90º convex and concave turns and 180º convex turns. Figures 5.8 and 5.9 below present the mock-up area with two rooms connected to the main area. The room openings or doors are set to about 80-100cm wide, which is about the actual size of a real door opening.

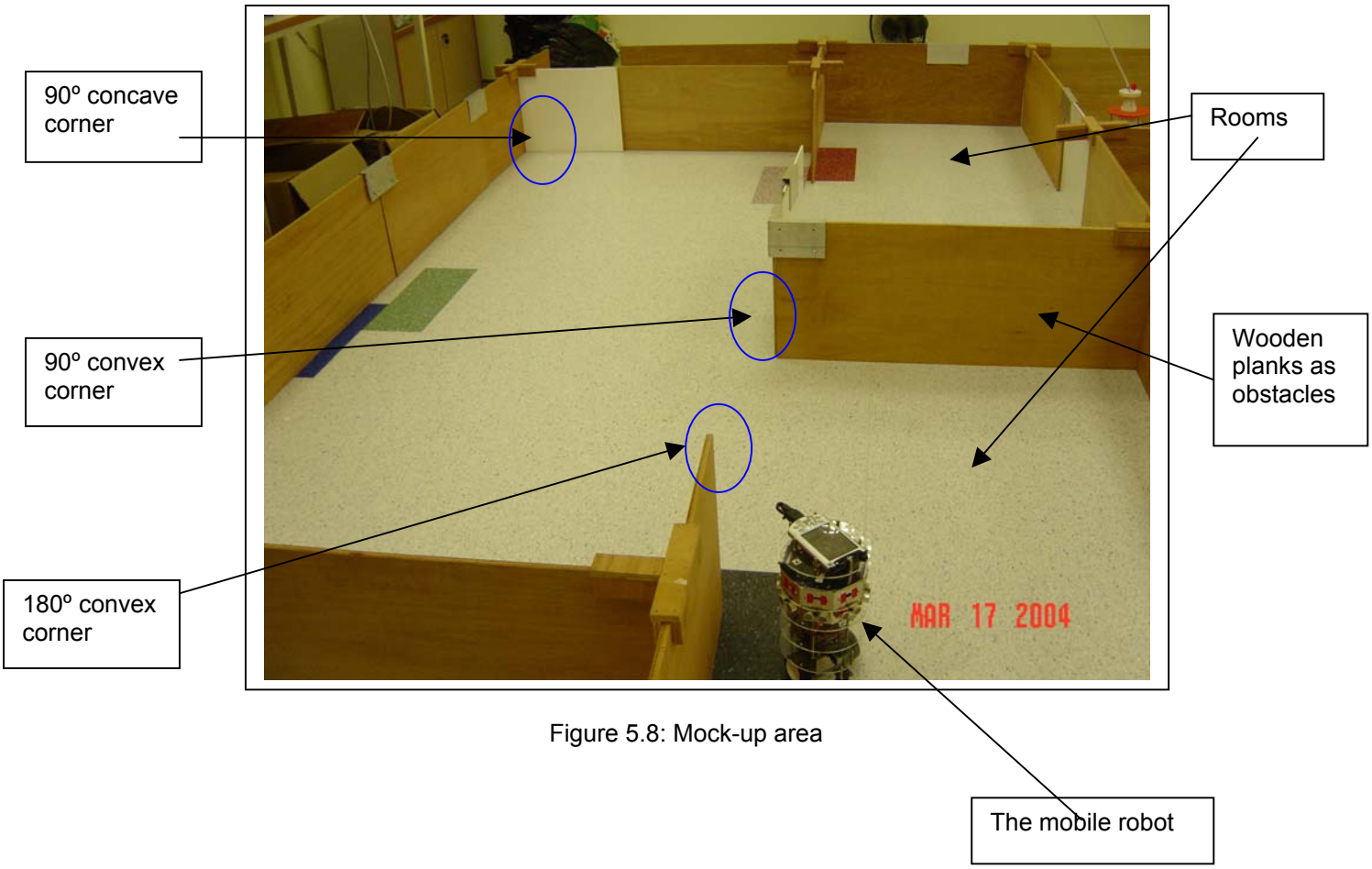


Figure 5.8: Mock-up area

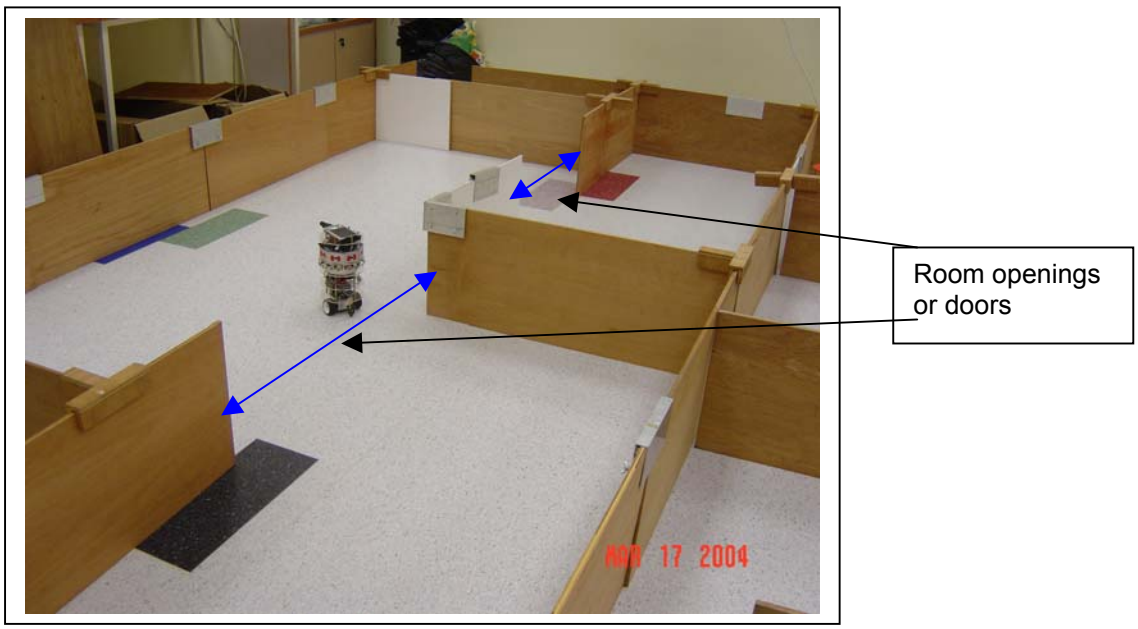


Figure 5.9: Mock-up area (another view)

The search however is conducted in a dark room so that the robot's light sensors are able to pick up the light intensities more efficiently. Initial experiments with the photoelectric light sensors have shown that they could not be relied upon to distinctly differentiate the light intensities from the fluorescent light bulb (target) and the lit-up room. Figure 5.10 shows a light source in the same layout in a dark room.

Figure 5.10: Light source in dark room

5.4 Light sensors' considerations

The light sensors on the robot are enclosed in an insulated rubber tube (please see figure 5.11) so that to make it less sensitive to the reflections caused by the illuminated light source. The two search algorithms work in such a way that when a set light intensity reading is picked up, it will perform the search function. Therefore it is important that the light sensors do not detect "wild" intensity readings, which do not belong to the actual light source.

This would prevent the consequences of the robot detecting a “fake” light source from occurring.

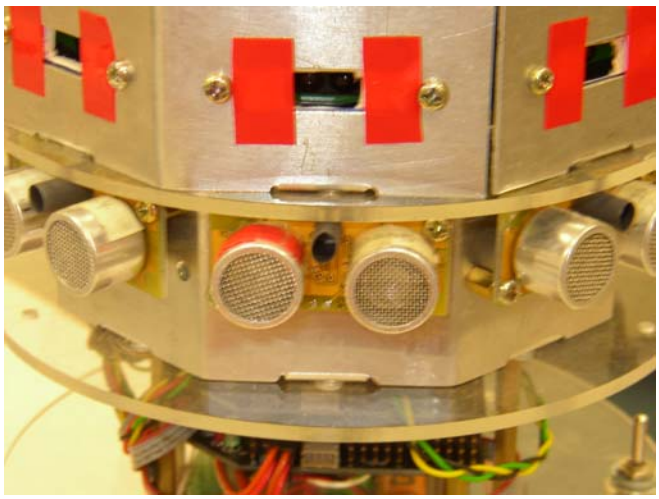


Figure 5.11: Enclosed light sensors in rubber tube casing

In the search function, the robot is programmed to be able to detect the light source from 1m away. Pre-experimental light sensor configuration tests thus has to done to detemine the detected intensity value for each sensor at 1m from the light source in a dark room. There are altogether 8 light sensors surrounding the robot as shown in figure 5.12. The yellow circular symbols denote the light sensors used to detect the light source, whereas the black ones are not used. Table 5.1 shows the light sensors’ setting used in both algorithms in the detection of the light source.

Table 5.1: Recorded settings of light sensors at *1m* from light source

Sonar sensor no.	Sensor intensity reading	Setting used in algorithms
224	114	114
226	78	78
228	104	104
238	184	184
236	159	159

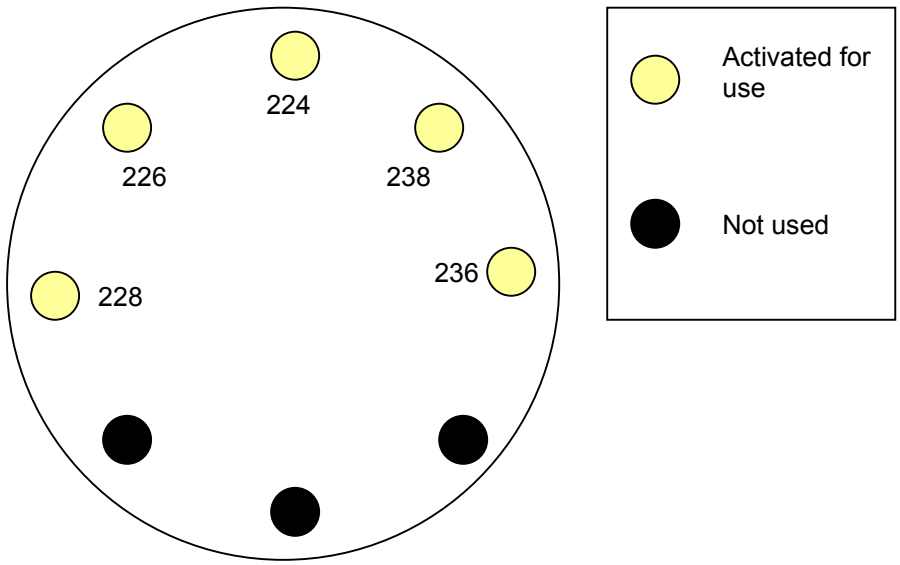


Figure 5.12: Light sensors' layout on the robot

5.5 Experimental Algorithm 1 (The wall-following search)

In the hardware experiment, the robot is made to travel slow at about 0.045 m/s while following the wall. The slow speed is even more important and necessary in real-time experiments because of the unpredictability nature of the sonar sensors, which have tendencies to return inaccurate values to the program. Thus sufficient leeway must be considered to give the robot enough time to veer away from the wall when it is too close to it. The hardware program is as described in section 3.3.

5.6 Experimental Algorithm 2 (The "most-open-area" search)

Though efficient in simulations, this algorithm presents some complexity during the hardware tests. First of all, the "south, "south-west" and "south-east" decisions (when the robot is "stuck" at a wall) are omitted because the robot is likely to detect its previous path and thus senses the same open area

which it has just come out from. In the simulations, 2-D straight-line sonar rays are used and the “south” decision is omitted because of the same reason. The explanation behind this is due to the wide angular coverage of the 3-D cone-shape sonar pulse of the real sensor. The “east” and “west” directions in the decision function in the hardware tests provide sufficient coverage as these directions also overlap into the “south-east” and “south-west” directions. Since this algorithm does not have the wall as a guide, it has its own obstacle avoidance ability such that it tries not to crash into the walls when it is too near. It is also important to fine-tune the robot such that it is able to travel in a straight line like in simulations while in the *move* function. A straight-line speed of about 0.09m/s is given to the robot in this case as it can afford to move faster as does not have to use the wall as a guide.

5.7 Hardware experimental methods

Similar to the simulation tests, 14 different combinations of robot and light source target positions are used in the hardware tests (same as the combinations illustrated in section 4.5). A stopwatch is used to time the two search methods and the algorithm ends when the robot comes to a complete halt within 30cm from the light source. Like in the simulation tests, 5 experiment tests are performed to generate an average time taken value for each combination and an experiment “failure” is said to occur after 12 minutes of futile search. That would imply that the robot took too long to search for the light source.

5.8 Analysis of hardware test results

The detailed results of the hardware experiments have been compiled and can be viewed in appendix 9.6.

Table 5.2: Tabulated summarised results of hardware experiments

Combination No.	Average time-taken for search		Success Rate (out of 5 tests)	
	Wall-following	"Most-open-area"	Wall-following	"Most-open-area"
1	3 mins 34 secs	4 mins 27 secs	100%	80%
2	5 mins 21 secs	1 min 40 secs	100%	100%
3	6 mins 52 secs	42 secs	100%	100%
4	3 mins 49 secs	5 mins 16 secs	100%	80%
5	2 mins 58 secs	4 mins 26 secs	100%	80%
6	1 min 15 secs	1 min 7 secs	100%	60%
7	1 min 23 secs	1 min 0 secs	100%	100%
8	5 mins 59 secs	1 min 27 secs	100%	100%
9	6 mins 56 secs	2 mins 46 secs	100%	100%
10	6 min 56 secs	44 secs	100%	100%
11	3 mins 15 secs	4 mins 12 secs	100%	60%
12	1 min 23 secs	2 mins 55 secs	100%	100%
13	1 min 20 secs	45 secs	100%	100%
14	2 mins 23 secs	57 secs	100%	100%

The summarised results from hardware experiments displayed in table 5.2 have once again highlighted the wall-following search is extremely efficient in the sense that it gives a 100% success rate in the experiments. This of course can be attributed to the inherent consistency of wall-following an environment, which can hardly go wrong. Similarly, the success rates from the "most-open-area" search tests were not perfect and once again shows its vulnerability in this aspect. However, this comes as no surprise as hardware sonar sensors' 3-D sensing pulses are not perfect if used to detect raw values. Also, depending on the relative position of the robot to the room openings, the robot may or may not be able to detect a room opening (this can be explained in section 5.9)

In any case, the latter search algorithm still manages faster search times in the majority of the combinations just like in the simulated tests. This can be explained by the faster straight-line speed used and the method's ability to move the robot quickly across open areas.

However, it can also be seen that the wall-following search provides slightly faster times in a couple of the different combinational tests. This could be due to the lack of speed efficiency on the part of the other search method in these tests.

5.9 Discrepancies in the types of sonar sensors

The inconsistencies and unparalleled relationship between the results of the simulations and hardware tests of the "most-open-area" search are likely to be caused by the differences in the types of sonar sensors used. In the *Player & Stage* simulations, the sonar rays are 2-D and hence 3 rays are grouped together as "one" sonar sensor to represent the sensing of one direction in the "most-open-area" search algorithm. This is better illustrated in the diagram shown in figure 5.13 below.

The sum of the rays (in green) of sp[0], sp[1] and sp[2] is represented by sonar sensor "224" and the sum of the rays of sp[23], sp[22] and sp[21] is represented by sonar sensor "238" respectively. For an example, if the robot is to make a decision here as to which is the "most-open-area", the simulation decision-making function will decide that the "north" direction (sonar "224") is more open. One the other hand, in the hardware tests, there exist

unpredictability such that the “most-open-area” being returned could be either the “north” or “north-east” direction.

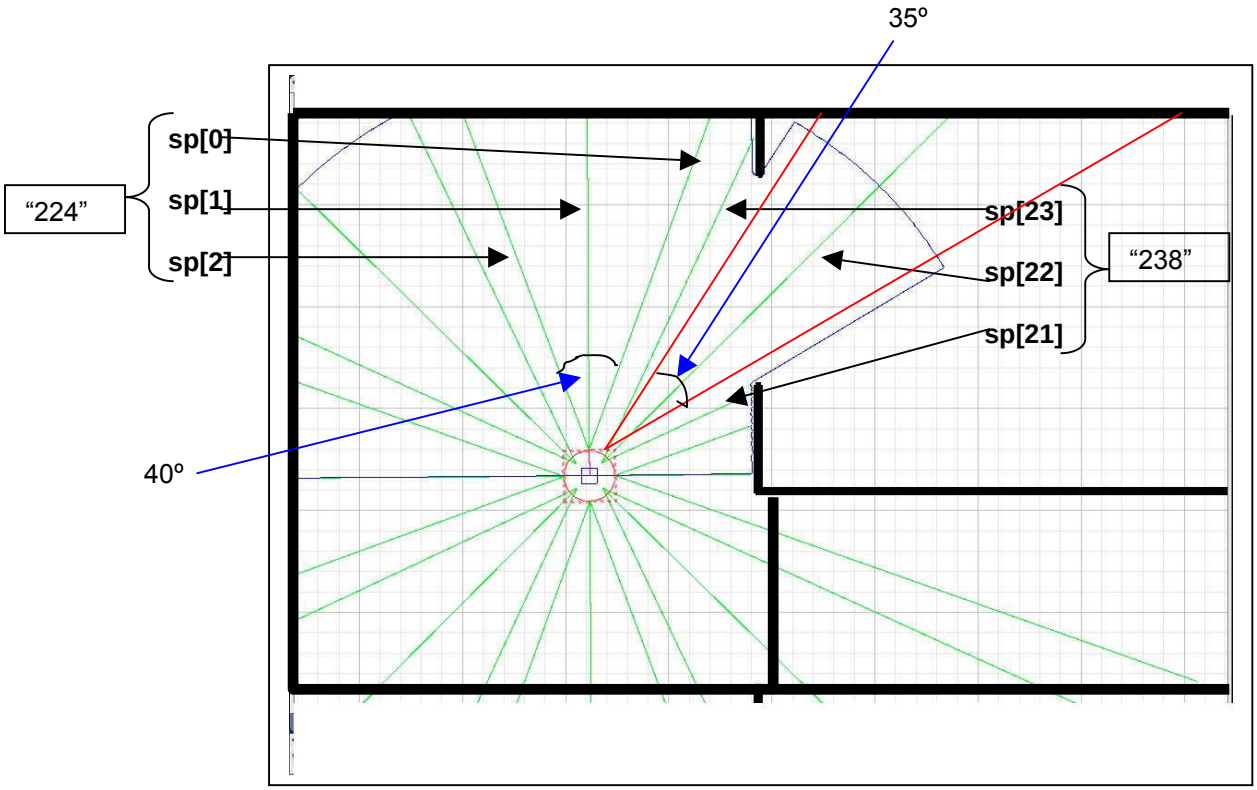


Figure 5.13: Grouping of 2-D sonar rays

The reason is because the actual sonar sensor though may have a scanning range of 40º, but due to its unpredictability, it could sometimes return a scanning range of maybe say 35º, as shown in the red lines in the diagram. If this is the case, the hardware program will return the “north-east” direction as the “most-open-area”.

The hardware sensors contribute another set of problem with its 3-D conical-shape scanning radius and this is illustrated in figure 5.14 below. When the robot is near to the opening of a room for an example, its sonar sensor, represented by the blue-coloured rays is able to detect the open space ahead because its coverage radius does not coincide with the walls. The red-

coloured rays represent the sonar's coverage of a robot further away from the wall and contrary to the previous scenario, the robot now detects the wall as an obstruction and hence is not able to sense the open space ahead. This problem is related to the "most-open-area" search method when applied to actual hardware experimentation.

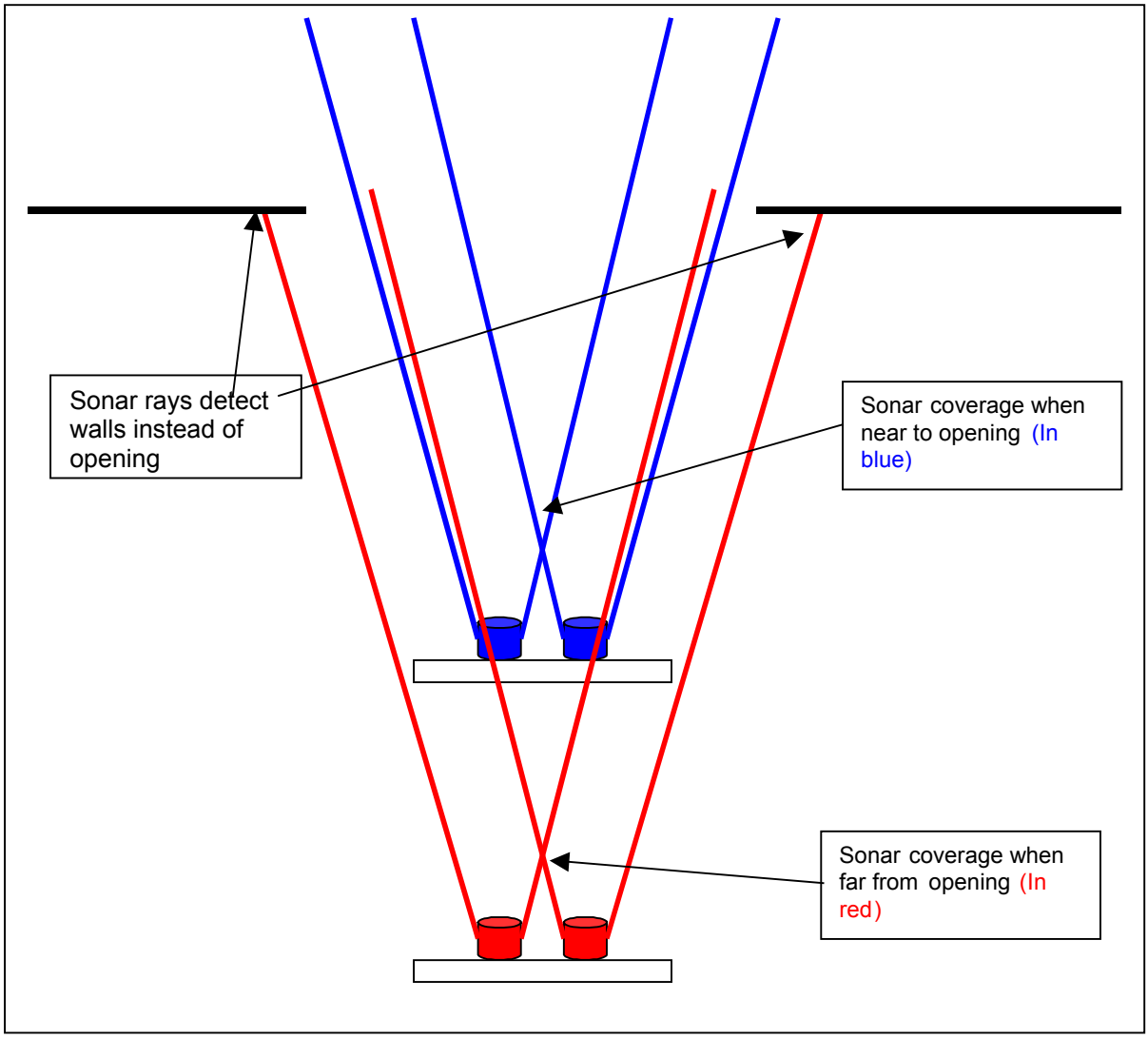


Figure 5.14: Hardware sonar discrepancy

Chapter 6: Conclusion

Two search algorithms have been developed to search for a light target source in a flat or office-like environment. They have also been tested and compared both in simulations and hardware experiments. Table 6.1 summarises the characteristics of the two algorithms from the simulation and hardware combined test results.

Table 6.1: Summary of characteristics of the two search algorithms

	Wall-following search	**"Most-open-area" search**
Success rate *(Able to complete under 12 minutes)*	-Extremely consistent	-Less consistent (About 88% successful)
Time taken to complete search	-Relatively slower	-Relatively faster (About 71% of the time)
Remarks	-Extremely reliable in finding target -Should be used if time is not an important factor in the search	- Should be used during urgency and when willing to take some risks - Less reliable in finding target

Hence, the project has shown that the two search algorithms are effective in different ways and they should be adopted for use according to the needs and purposes of the search.

Simulation and hardware results both reaffirmed that the wall-following search was very consistent in terms of success rate. Results also showed that the "most-open-area" search algorithm was able to complete most of the search combination tasks in a faster time

The results showed that the wall-following search ranked higher in terms of success efficiency and the “most-open-area” search, though not as consistent, provided a faster search method in such an environment. Some recommendations have been made on how to further improve these algorithms.

Chapter 7: Further discussion and recommendations

7.1 Further discussion on wall-following search

The wall-following search method has proved to be a very reliable and meticulous search method in such a flat or office-like environment, with fixed right angle or 180º corners. Such a method will also work effectively for a more complex office-like environment with numerous rooms and corridors. The only drawback in this case is the search may take a very long time to complete if the target is in the end corner of layout. One way to solve this problem would be to increase the wall-following speed, which will require better sensors to feedback fast and reliable information so that the robot would still be able to follow the wall without compromising on crashing against the walls

7.2 Further discussion on “most- open-area” search

This search method will probably not work as well in an extended flat or office-like environment because it may never move into an “open” area if its sensors are not able to detect the room openings. However, this approach is likely to work in layouts with more varieties of angles and curvatures, that is non-flat or office-like environments because it does not rely on specific corners to turn. On further modification and improvement, it could also work in layouts with different obstacles lying within.

7.3 Recommendations

1) Without time constraints, more tests can be conducted using the two search algorithms in different flat or office-like layouts, whether they are simpler or more complex ones. That will provide more information for comparison and analysis.

2) To better correlate the simulations with the hardware tests and minimise the discrepancy of the sonar sensors, 3-D simulations should be employed. A 3-D GUI robot software called *gazebo* can be used instead of the 2-D *Stage*. During the designing period of these two algorithms, *gazebo* was not fully implemented for use and contained bugs.

3) Employing the use of laser rangefinders in a combinational way can lessen the unpredictability of the sonar sensors. The laser rangefinder works in a 2-D plane, and is far more accurate than the 3-D conical projection of the sonar. Thus, with this combination, uncertainties due to specular reflections and inaccuracies due to unnecessary detections of edges or even the ground can be reduced. Such a combination can also be used in the detection and recognition of more varieties of landmarks and obstacles, thus providing the algorithms with more dimensions and flexibility.

4) Odometry will provide another huge dimension to designing the algorithms. In a simple robot platform, the inaccuracies arising from the use of odometry can probably be minimised if different types of wheel encoders. For example, the rate encoder and the mechanical encoder can be deployed for use

together providing there is a method to resolve their different outputs and combining them to a reliable output source.

If such means cannot be met, localised use of a simple odometry method can also help to improve the search method. For example, the robot deploys the use of odometry only when there is a need to circle an obstacle or to retrace its steps after entering a room. The inaccuracies from wheel slippages can be reduced to a minimum if the robot travels only short distances.

Chapter 8: References

[1] Keith :. Doty and Steven Louis Seed. Autonomous Agent Map Construction in Unknown Enclosed Environments. MLC-COLT ’94 Robot Learning Workshop, Rutgers, New Brunswick, N.J.

[2] Brian Yamauchi, Navy Centre for Applied Research in Artificial Intelligence, Naval Research Laboratory, Washington DC 20375-5337. A Frontier-Based Approach for Autonomous Exploration.

[3] M. Khatib, B. Bouilly, T. Siméon and R. Chatila. Indoor Navigation with Uncertainty using Sensor-based Motions. IEEE Int. Conf. On Robotics and Automation, Albuquerque, USA, April 20-25, 1997.

[4] Teruko YATA, Lindsay KLEEMAN and Shin’ichi YUTA. Wall Following Using Angle Information Measured by a Single Ultrasonic Transducer. IEEE Int. Conf. On Robotics & Automation, Leuven, Belgium, May 1998.

[5] P. New man, M. Bosse and J. Leonard, Massachusetts Institute of Technology. Autonomous Feature-Based Exploration.

[6] Maxim A. Batalin and Gaurav S. Sukhatme. Efficient Exploration without localisation. Intl. Conference on Robotics and Automation (ICRA2003), Taipei, Taiwan, May 12-17, 2003.

[7] Robert Grabowski, Pradeep Khosla and Howie Choset. Autonomous Exploration via Regions of Interest. IEEE/RSJ Int’l Conference on Intelligent Robots and Systems, Las Vegas, Neveda, October 2003.

[8] Erol Gelenbe, Nestor Schmajuk, John Staddon and John Reif. Autonomous search by robots and animals: A survey. Robotics and Autonomous Systems 22 (1997) 23-34.

[9] Robin R.Murphy. Introduction to AI Robotics (Intelligent Robotics and Autonomous Agents).

[10] Brian P. Gerkey, Richard T. Vaughan, Kasper Støy, Andrew Howard, Maja J Matarić and Gaurav S Sukhatme. Most Valuable Player: A Robot Device Server for Distributed Control. In Proc. Of the IEEE/RSJ Intl. Conf. On Intelligent Robots and Systems (IROS), pages 1226-1231, Wailea, Hawaii, October 2001.

[11] Richard T. Vaughan. "Stage: A Multiple Robot Simulator". Technical Report IRIS-00-394, Institute for Robotics and Intelligent Systems, School of Engineering, University of Southern California, 2000.

[12] Acroname Inc. (http://www.acroname.com)

Appendices
(Chapter 9)

Appendix 9.1: Player & Stage

9.1.1 Player

Player is a multi-threaded robot device server, which gives simple and complete control over the physical sensors and actuators on the mobile robot. When Player is running on the robot, the client control program connects to it via a standard TCP socket, and communication is accomplished by the sending and receiving of small sets of simple messages. Client programs can be written in any language that can open and control a TCP socket. In this project, the C++ language is used to write client programs to establish connection with the server, which is the simulated Stage robot in this case. An example of the client/server interaction is illustrated in figure 9.1

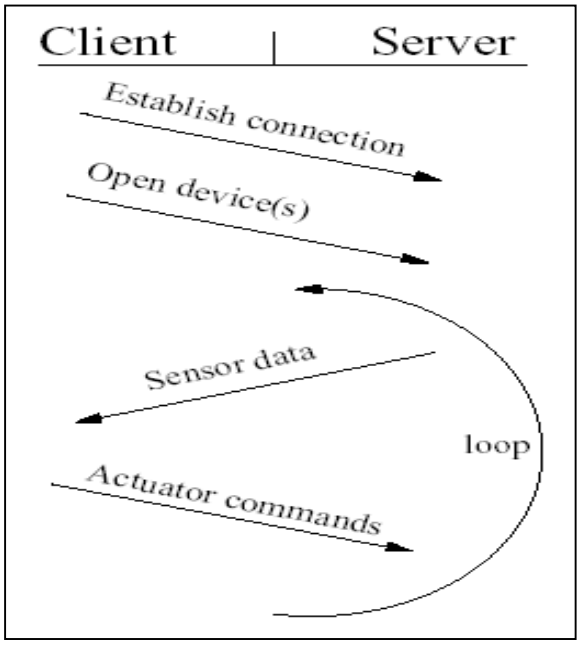


Figure 9.1: Example of the client/server relationship

Player is able to control different kinds of devices such as the sonar, laser, position and the GPS. When a client reads from a device, the client receives the device's current data. For a Player device, the term *data* is used to refer to

the salient state of the device. It is also important to note that the transmission speed of the devices' data to the client is usually 10 Hz.

A user must first create a *PlayerClient* proxy and use it to establish a connection to a Player server. Next, the proxies of the appropriate device-specific types are created and initialised before the actual algorithm code starts. A personalised *makefile* is also necessary to compile and *make* the program before they can actually be run. An example of a *makefile* is shown in figure 9.2

```
# location of libs & stuff
top_srcdir=/home/barch/fyp

CC = gcc
CPP = gcc -E
CXX = c++

CPPFLAGS =
LDFLAGS =
LIBS = -lpthread -ldl -lnsl
OBJ = jf.o

INCLUDES= -I$(top_srcdir)/include
LDADD = -L$(top_srcdir)/lib -lplayerclient

CXXFLAGS = -g -O2
CXXCOMPILE = $(CXX) $(INCLUDES) $(CXXFLAGS)
CXXLD = $(CXX)
CXXLINK = $(CXXLD) $(AM_CXXFLAGS)
$(CXXFLAGS) $(LDFLAGS) -o $@

# all is the first option in the makefile
# therefore is you just call 'make', it will assume you want
'make all'
.SUFFIXES:
.SUFFIXES: .S .c .cc .o .s
all: jf

jf: $(OBJ)
        $(CXXLINK) $(OBJ) $(LDADD) $(LIBS)

# can never remember what $< means, but use it
.c.o:
        $(COMPILE) -c $<
.cc.o:
        $(CXXCOMPILE) -c $<

# useful bits to clean & rebuild the code
clean:
        rm -f *.o *.so *.a
        rm -f jf

rebuild: clean all
```

Figure 9.2: The *makefile*

9.1.2 Player C++ Client

The Player C++ Client library is built on a "service proxy" model to hold two kinds of proxies for remote services. The Player Client proxy is created to establish a connection to a Player before other device-specific types of proxies (classes) can be created to link up with it. The developers of Player

mentioned that this library is not perfect and not straightforward but should be usable for anyone with knowledge of the C++ language. When the Player Client proxy is created, a default constructor is used to connect it to the server listening at *localhost: 6665*

The Player Client proxy

The most important function inside this class is the Read(). This method reads one round of data for each device and these data are processed by appropriate device proxies and stored there to be accessed by the program. For example, the partial code shown in figure 9.3 illustrates the function of the Read().

```
/* Go into read-think-act loop */
   while (stuck)
   {
            if(robot.Read())
            exit(1);

            /*Code can be written here*/
   }
```

Figure 9.3: The *Read ()* in Player Client

The PositionProxy

This class is used to control the position device, which in this case is the mobile robot. The motor command is called using this proxy to give the robot its speed and turn rate in mm/s and degrees/s respectively. Some of its attributes and methods used in this project are shown in figure 9.4.

Attributes
int xpos,ypos,theta;
Robot pose (according to odometry) in mm, mm, degrees.

Methods
PositionProxy(PlayerClient* pc, unsigned short index,
unsigned char access =’c’)
Constructor. Leave the access field empty to start unconnected.

int SetSpeed(int speed, int turnrate)
Same as the previous SetSpeed(), but doesn’t take the sideways speed (so use this one for non-holonomic
robots).

int ResetOdometry()
Reset odometry to (0,0,0). Returns: 0 if everything’s ok, -1 otherwise.

void Print()
Print current position device state.

Figure 9.4: The Position Proxy class reference

The SonarProxy

Sonar devices can be controlled via the SonarProxy class as it stores all necessary sonar range data. Thus range data can be accessed and utilised for obstacle avoidance for example.

Attributes
unsigned short range_count;
The number of sonar readings received.

unsigned short ranges
[PLAYER_SONAR_MAX_SAMPLES];
The latest sonar scan data. Range is measured in mm.

Methods
SonarProxy(PlayerClient* pc, unsigned short index,unsigned char access = ’c’)
Constructor. Leave the access field empty to start unconnected.

int SetSonarState(unsigned char state)
Enable/disable the sonars. Set state to 1 to enable, 0 to disable. Note that when sonars are disabled the client will still receive sonar data, but the ranges will always be the last value read from the sonars before they were disabled. Returns 0 on success, -1 if there is a problem.

void Print()
Print out current sonar range data.

Figure 9.5: The Sonar Proxy class reference

The LaserProxy class is primarily used to scan for the “light” source. In this project, the “light” source is actually a simulated object, which is “visible” to the laser device. Thus, this class allows laser range and intensity data be accessed directly. Some of its attributes and methods used are shown in figure 9.6.

Attributes
short min_angle, max_angle;
Scan range for the latest set of data. Angles are measured in units of 0.01º, in the range -9000 (-90º) to +9000 (+90º).

unsigned short resolution;
Scan resolution for the latest set of data. Resolution is measured in units of 0.01º. Valid values are: 25, 50, and 100.

bool intensity;
Whether or not reflectance (i.e., intensity) values are being returned.

unsigned short ranges
[PLAYER_LASER_MAX_SAMPLES];
The range values (in mm).

Methods
LaserProxy(PlayerClient* pc, unsigned short index,
unsigned char access=’c’)
Constructor. Leave the access field empty to start unconnected.

int Configure(short min_angle, short max_angle, unsigned short resolution, bool intensity)
Configure the laser scan pattern. Angles min_angle and max_angle are measured in units of 0.1º, in the range -9000 to +9000. Resolution is also measured in units of 0.1º. Valid values are: 25, 50 and 100. Set intensity to true to enable intensity measurements, or false to disable. Returns the 0 on success, or -1 of there is a problem.

int RangeCount ()
Get the number of range/intensity readings.

void Print()
Print out the latest laser range/intensity data.

Figure 9.6: The Laser Proxy class reference

9.1.3 Stage

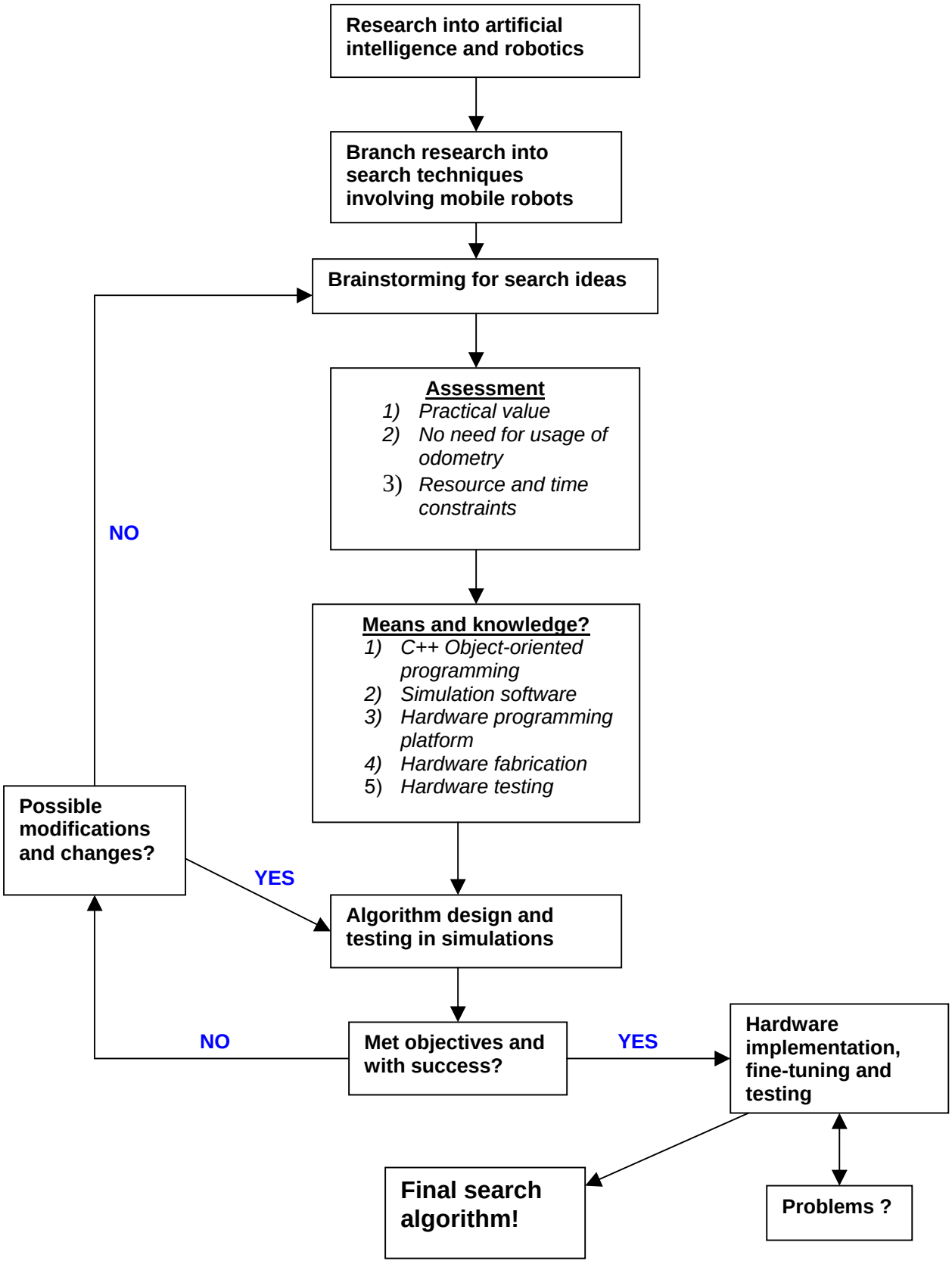
Stage is a collection of software tools to support the simulations of mobile robots, sensors and objects in a two-dimensional bitmapped environment. It is controlled through Player, which provides a conventional interface to a set of device drivers for real robots and sensors. Stage configuration files are called world files and they load up when Stage starts and effectively spawn Player as well. The *world files* describe the robot's specifications and definitions, sensors, actuators and objects. The virtual robots in Stage are controlled via Player in a client-server relationship. Figure 9.7 shows part of the *world file* used in this project.

Note: Player and Stage were developed jointly at the USC Robotics Research Lab and HRL Labs and are freely available under the GNU General Public License from http://playerstage.sourceforge.net.

```
# Description: 1 robot in a room layout, with sonars and laser scanner (to detect target)

# the resolution of Stage's raytrace model in metres
#
resolution 0.02

# GUI settings
#
gui
(
  size [ 505.000 505.000 ]
  origin [5 5 0]
  scale 0.021 # the size of each bitmap pixel in metres
  # This turns out to create an area of about 10.625m by 10.625m.
)

# load a bitmapped environment from a file
#
bitmap
(
  file "office.pnm.gz"
  resolution 0.02

)

# define an object representing a light source
box
(
  pose [8.759 0.737 0.000]
  shape "circle"
  #colour "green"
  size [0.5 0.5]
  laser_return "bright"
  sonar_return "visible"
  #fiducial_id 4
)

include "room.inc"
# This is an include file.

# create a round-shaped robot, and equipping it with 16 sonar scanners all round to simulate the sonar ability of
# the actual robot.

position
        (
        name "robot"
        port 6665
        shape "circle"
        pose [0.726 9.194 -86.797]
        size [0.5 0.5]

        p2dx_sonar(max_range 6.0)
        #laser((fiducialfinder())
        laser(max_range 4.0)
        truth()

        )
```

Figure 9.7: The Stage world file

Appendix 9.2: The algorithm design process (Flowchart)



Appendix 9.3: Simulation experimental results

Results for combination (1)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	3mins 3secs	2 mins 2 secs
2	3mins 35 secs	4 mins 22 secs
3	3mins 30 secs	2 mins 20 secs
4	3mins 23 secs	Fail (> 12 mins)
5	3mins 5 secs	3 mins 45 secs
Mean	3 mins 19 secs	3 mins 7 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	1
Success rate	100%	80%

Results for combination (2)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	6 mins 37 secs	1 min 30 secs
2	6 mins 30 secs	Fail (> 12 mins)
3	6 mins 20 secs	Fail (> 12 mins)
4	6 mins 32 secs	1 min 9 secs
5	6 mins 43 secs	3 mins 49 secs
Mean	6 mins 32 secs	2 mins 9 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	2
Success rate	100%	60%

Results for combination (3)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	8 mins 50 secs	Fail (> 12 mins)
2	8 mins 55 secs	Fail (> 12 mins)
3	8 mins 38 secs	Fail (> 12 mins)
4	8 mins 52 secs	2 mins 43 secs
5	8 mins 34 secs	5 mins 12 secs
Mean	8 mins 46 secs	3 mins 58 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	2
Success rate	100%	60%

Results for combination (4)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	7 mins 52 secs	2 mins 44 secs
2	7 mins 55 secs	2 mins 44 secs
3	7 mins 50 secs	2 mins 57 secs
4	7 mins 43 secs	5 mins 4 secs
5	8 mins 3 secs	2 mins 52 secs
Mean	7 mins 53 secs	3 mins 16 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Results for combination (5)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	5 mins 8 secs	2 mins 42 secs
2	5 mins 14 secs	Fail (> 12 mins)
3	5 mins 4 secs	10 mins 6 secs
4	5 mins 7 secs	8 mins 47 secs
5	5 mins 15 secs	Fail (> 12 mins)
Mean	5 mins 10 secs	7 mins 12 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	2
Success rate	100%	60%

Results for combination (6)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	3 mins 19 secs	Fail (> 12 mins)
2	3 mins 15 secs	Fail (> 12 mins)
3	3 mins 21 secs	1 min 23 secs
4	3 mins 20 secs	4 mins 55 secs
5	3 mins 25 secs	5 mins 44 secs
Mean	3 mins 20 secs	4 mins 1 sec

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	2
Success rate	100%	60%

Results for combination (7)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	2 mins 18 secs	1 min 32 secs
2	2 mins 19 secs	1 min 30 secs
3	2 mins 22 secs	1 min 32 secs
4	2 mins 10 secs	1 min 37 secs
5	2 mins 15 secs	1 min 30 secs
Mean	2 mins 17 secs	1 min 32 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Results for combination (8)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	4 mins 46 secs	1 min 46 secs
2	5 mins 3 secs	1 min 47 secs
3	4 mins 44 secs	1 min 45 secs
4	4 mins 51 secs	1 min 47 secs
5	4 mins 43 secs	1 min 47 secs
Mean	4 mins 49 secs	1 min 46 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Results for combination (9)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	7 mins 40secs	2 mins 12 secs
2	7 mins 48 secs	2 mins 14 secs
3	7 mins 35 secs	5 mins 14 secs
4	7 mins 40 secs	5 mins 37 secs
5	7 mins 35 secs	2 mins 10 secs
Mean	7 mins 40 secs	3 mins 29 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Results for combination (10)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	1 min 40secs	46 secs
2	1 min 40 secs	47 secs
3	1 min 36 secs	46 secs
4	1 min 45 secs	46 secs
5	1 min 35 secs	48 secs
Mean	1 min 39 secs	47 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Results for combination (11)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	4 mins 0 secs	1 min 12 secs
2	4 mins 4 secs	1 min 12 secs
3	3 mins 57 secs	1 min 25 secs
4	3 mins 55 secs	5 mins 47 secs
5	4 mins 1 sec	1 min 10 secs
Mean	3 mins 59 secs	2 mins 9 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Results for combination (12)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	1 min 10secs	2 mins 17 secs
2	1 min 11 secs	2 mins 12 secs
3	1 min 11 secs	2 mins 25 secs
4	1 min 12 secs	3 mins 47 secs
5	1 min 11 secs	4 mins 40 secs
Mean	1 min 11 secs	3 mins 4 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Results for combination (13)

No.	Time taken	
	Wall-following search	**"Most-open-area" search**
1	2 mins 24 secs	1 min 6 secs
2	2 mins 27 secs	1 min 7 secs
3	2 mins 27 secs	1 min 10 secs
4	2 mins 25 secs	1 min 7 secs
5	2 mins 34 secs	1 min 10 secs
Mean	2 mins 27 secs	1 min 8 secs

	Wall-following search	**"Most-open-area" search**
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Results for combination (14)

No.	Time taken	
	Wall-following search	**"Most-open-area" search**
1	2 mins 50secs	2 mins 27 secs
2	2 mins 51 secs	1 min 42 secs
3	2 mins 47 secs	1 min 20 secs
4	2 mins 55 secs	1 min 17 secs
5	2 mins 50 secs	1 min 15 secs
Mean	2 mins 51 secs	1 min 36 secs

	Wall-following search	**"Most-open-area" search**
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Appendix 9.4: The Brainstem technology

The Architecture

The Brainstem architecture provides a standardised interface between conventional computer hardware such as personal computers, PDAs and the real world needs of robots including motor control and sensor inputs.

Its architecture as shown below in figure 9.8 allows the impartation of artificial intelligence in rich computer environments to specific hardware. In addition, the cross-platform ability of the architecture allows higher-level software constructs, algorithms, and development easily.

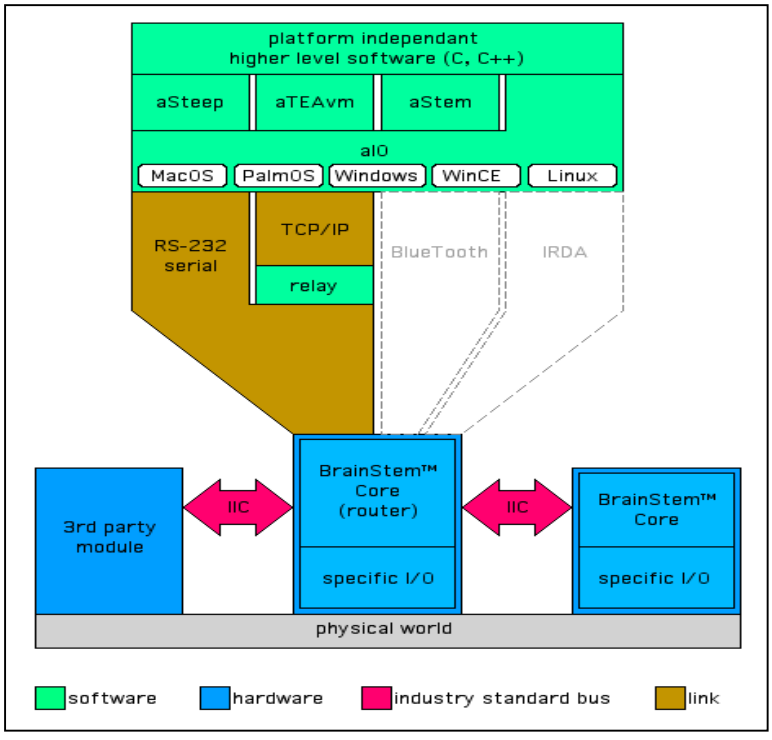


Figure 9.8: Architecture of Brainstem technology

The Link and the GP 1.0 module

The link is an abstract connection between the software support libraries and the physical Brainstem module. Currently this link is in the form of a serial

connection. The software libraries can be switched between serial, TCP/IP, USB, Blue Tooth or other protocol layers as they are supported. The protocol across the link is in the form of packets, each of which has a maximum limit of 8 bytes. The module (mounted on the robot) communicates with the host (e.g. the Pocket PC) across the link and it has a common bus interface that uses the 2-wire IIC (Information Integration Centre) protocol.

The GP 1.0 module

The GP 1.0 module provides a variety of I/O functions for robotics and control systems. Combining programmability and a built-in command set, this module is both a stand-alone controller and real-world interfacing tool.

Some of the capabilities, which the module provides, are:

- Onboard 1 A, 5V power regulation (low dropout)
- 40 MHz RISC processor
- 4 servo outputs with 8-bit resolution, variable range, and speed control
- 5 10-bit resolution A/D inputs
- 5 digital I/O pins with polling and timing functions
- 4 pins can be used for logic event timing
- 1MB IIC port
- RS-232 TTL serial port
- Sharp GP2D02 driver
- 544 bytes of RAM available to user
- Small size (2.5" square, 0.5" high) and easily stacked

Figure 9.9 shows how the GP 1.0 module chip looks like.

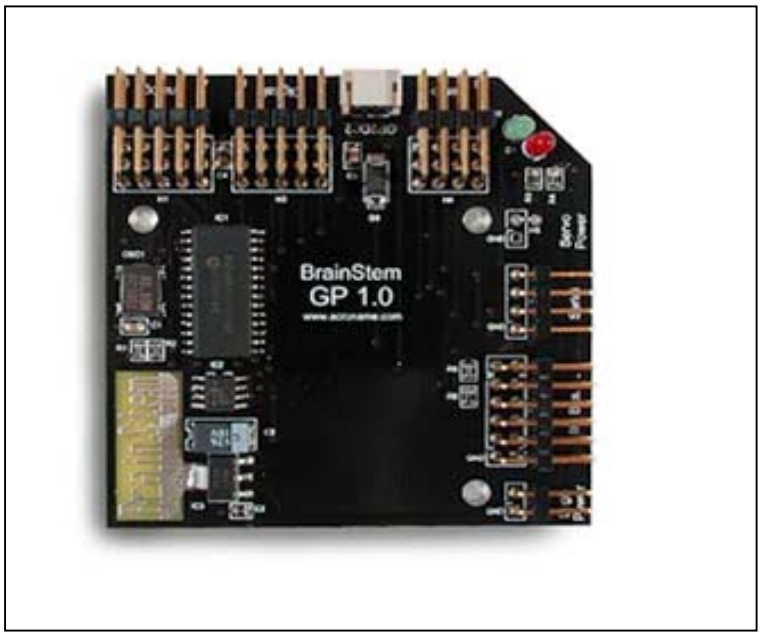


Figure 9.9: The GP 1.0 Brainstem module

Power and Voltage Ratings

The recommended input voltage for the GP 1.0 module onboard voltage regulator is 4.5V to 12V. At higher voltages within this range, the regulator will generate a lot of heat and the heatsink may get quite warm. The regulator is a LDO (low drop-out) LM2940 so it begins to regulate the current at about 5.5V and up. Below that, the regulator acts more like a straight pass-through with little regulation. The voltage regulator has an output current rating of 1 A. Normally, the GP 1.0 module uses 50mA at 6V.

Power Supplies

The Brainstem GP 1.0 module has two power supply inputs, one for logic and one for servos. A 9V battery is used as the logic power supply for the GP module. An insufficient power supply can create low voltage conditions, which will make the processor, reset or malfunction. Servos have a secondary

connector for power and the usual operating voltage for servos is 6V. A pin-out diagram of the GP 1.0 module is shown below in figure 9.10.

Processor

The processor on the module is a Microchip PIC18C252 and it runs at 40MHz. The module also includes a Microchip 24FC128/256 EEPROM for storage of system data, reflex commands, and VM opcodes. This EEPROM has an access rate of 1Mbit/sec. The processor has IIC interface hardware that has a maximum data transfer rate of 1Mbit/sec.

Command Reference

When the module receives a command, it responds by performing a simple I/O or system task such as moving the servos. Commands are sent as packets of binary bytes. The first byte is the command code and the subsequent bytes are data for the command. Commands can be issued from the host using the C or C++ language.

Appendix 9.5: The Sensors

The Acroname Devantec SRF08 RangeFinder sensor is used in this project. Prior to actual hardware testing, some experimental calibration was carried out to verify the accuracies of the sensors. Each sonar sensor was found to project a 3-D cone-shape sound pulse of about 40º, and thus two of such sensors mounted together would give project a sound pulse area as shown in figure 9.11 below. Note that there is a blind area in between the two sensors mounted on the manufactured chipboard. Further tests also showed that the maximum range of the sonar sensor is not 6m, but actually about 1.1m.

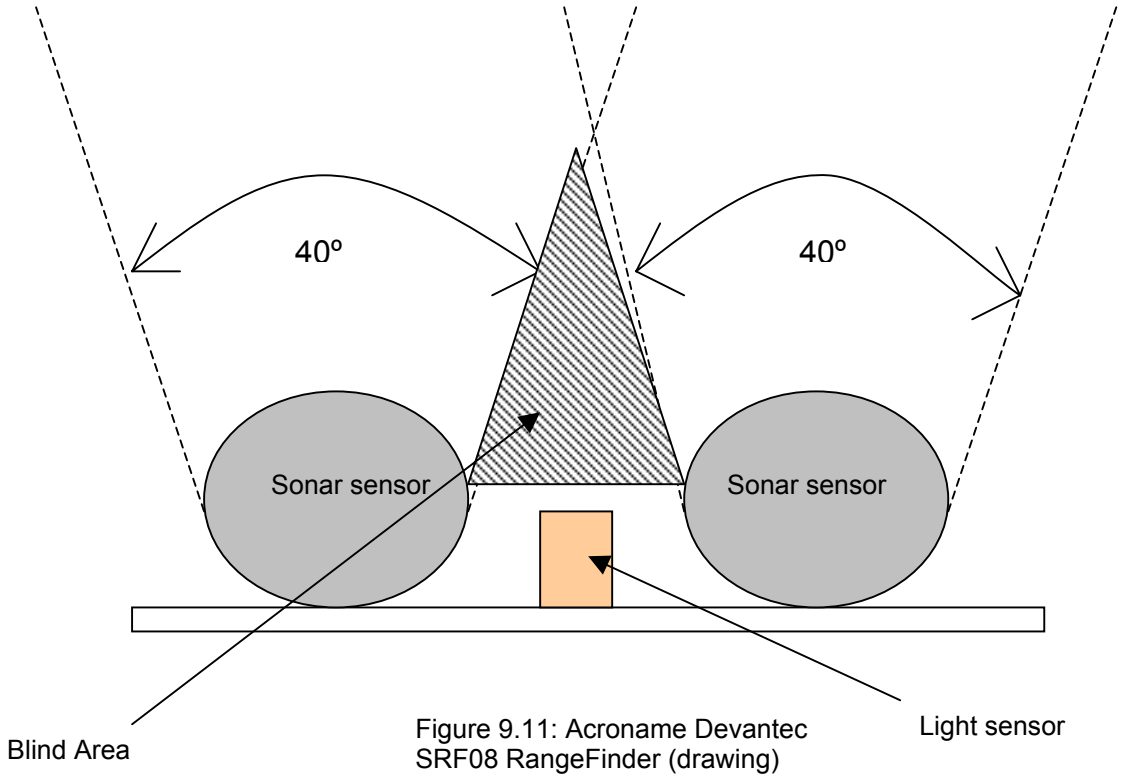


Figure 9.11: Acroname Devantec SRF08 RangeFinder (drawing)

Appendix 9.6: Hardware experimental results

Results for combination (1)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	3mins 30secs	5 mins 2 secs
2	3mins 40 secs	4 mins 32 secs
3	3mins 14 secs	5 mins 20 secs
4	3mins 23 secs	2 mins 55 secs
5	4mins 5 secs	Fail (> 12 mins)
Mean	3 mins 34 secs	4 mins 27 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	1
Success rate	100%	80%

Results for combination (2)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	5 mins 0 secs	52 secs
2	5 mins 4 secs	57 secs
3	5 mins 23 secs	1 min 28 secs
4	5 mins 46 secs	1 min 36 secs
5	5 mins 30 secs	3 mins 29 secs
Mean	5 mins 21 secs	1 min 40 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Results for combination (3)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	7 mins 0 secs	21 secs
2	6 mins 55 secs	24 secs
3	6 mins 38 secs	28 secs
4	6 mins 32 secs	1 min 47 secs
5	7 mins 14 secs	28 secs
Mean	6 mins 52 secs	42 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Results for combination (4)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	3 mins 36 secs	4 mins 44 secs
2	4 mins 15 secs	5 mins 29 secs
3	3 mins 50 secs	Fail (> 12 mins)
4	3 mins 43 secs	4 mins 14 secs
5	3 mins 39 secs	6 mins 35 secs
Mean	3 mins 49 secs	5 mins 16 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	2
Success rate	100%	80%

Results for combination (5)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	2 mins 56 secs	5 mins 32 secs
2	2 mins 57 secs	1 min 50 secs
3	3 mins 3 secs	9 mins 3 secs
4	2 mins 50 secs	Fail (> 12 mins)
5	3 mins 5 secs	1 mins 19 secs
Mean	2 mins 58 secs	4 mins 26 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	1
Success rate	100%	80%

Results for combination (6)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	1 min 19 secs	46 secs
2	1 min 8 secs	43 secs
3	1 min 21 secs	1 min 53 secs
4	1 min 15 secs	Fail (> 12 mins)
5	1 min 10 secs	Fail (> 12 mins)
Mean	1 min 15 secs	1 min 7 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	2
Success rate	100%	60%

Results for combination (7)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	1 min 26 secs	49 secs
2	1 min 18 secs	54 secs
3	1 min 20 secs	1 min 37 secs
4	1 min 25 secs	56 secs
5	1 min 25 secs	46 secs
Mean	1 min 23 secs	1 min 0 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Results for combination (8)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	6 mins 20 secs	56 secs
2	6 mins 13 secs	1 min 32 secs
3	5 mins 44 secs	2 min 16 secs
4	5 mins 59 secs	1 min 10 secs
5	5 mins 41 secs	1 min 23 secs
Mean	5 mins 59 secs	1 min 27 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Results for combination (9)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	6 mins 40secs	42 secs
2	6 mins 58 secs	3 mins 23 secs
3	7 mins 2 secs	3 mins 46 secs
4	6 mins 52 secs	1 min 25 secs
5	7 mins 10 secs	4 mins 34 secs
Mean	6 mins 56 secs	2 mins 46 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Results for combination (10)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	7 mins 1sec	39 secs
2	6 mins 49 secs	37 secs
3	6 mins 56 secs	38 secs
4	7 mins 9 secs	1min 6 secs
5	6 mins 45 secs	39 secs
Mean	6 min 56 secs	44 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Results for combination (11)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	3 mins 11 secs	Fail (> 12 mins)
2	3 mins 14 secs	3 mins 39 secs
3	3 mins 15 secs	Fail (> 12 mins)
4	3 mins 25 secs	4 mins 47 secs
5	3 mins 11 sec	4 mins 11 secs
Mean	3 mins 15 secs	4 mins 12 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	2
Success rate	100%	60%

Results for combination (12)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	1 min 22 secs	4 mins 47 secs
2	1 min 20 secs	5 mins 5 secs
3	1 min 20 secs	3 mins 13 secs
4	1 min 32 secs	47 secs
5	1 min 23 secs	43 secs
Mean	1 min 23 secs	2 mins 55 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Results for combination (13)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	1 min 24 secs	1 min 23 secs
2	1 min 17 secs	37 secs
3	1 min 19 secs	30 secs
4	1 min 25 secs	37 secs
5	1 min 14 secs	40 secs
Mean	1 min 20 secs	45 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Results for combination (14)

No.	Time taken	
	Wall-following search	"Most-open-area" search
1	2 mins 23secs	43 secs
2	2 mins 23 secs	41 secs
3	2 mins 27 secs	1 min 35 secs
4	2 mins 20 secs	41 secs
5	2 mins 20 secs	1 min 4 secs
Mean	2 mins 23 secs	57 secs

	Wall-following search	"Most-open-area" search
No. of tests	5	5
No. of failures	0	0
Success rate	100%	100%

Appendix 9.7: Problems encountered with *Player & Stage* (P/S)

1) There was a build error in the installation of Player 1.3.2 and Stage 1.3.2. Acoustics need to be disabled before the program can be compiled.

2) The source code of rtk_canvas_get_image_rgb16() in librtk/rtk_canvas.c needed modifications and debugging before video capture of simulations can be used.

3) The update cycle in Stage is slow, and resulted in the robot not responding fast enough to avoid obstacles in some instances. Sometimes the lag can be so bad that it might be worse than the actual hardware robot.

4) There were occasional times when the client programs consume data more slowly than the server is delivering it. After the TCP receive and send queues fill up, bytes start getting thrown away. This resulted in error messages such as, *"WARNING: server's message is too big (58392580 bytes). Truncating data."*

5) There were problems with the TruthProxy client when it was used initially to make the robot turn a specific angle. Experiments showed that if the robot was placed in a certain quadrant (faced in a certain way), it was not able to turn the angle it was asked to do so. It was suspected that the simulator took an unexpectedly long time to execute the changes.