DEVELOPMENT OF A VACUUM CLEANER MOBILE ROBOT

by

Alp Sardağ

B.S. in Computer Engineering, Boğaziçi University, 1996

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

Boğaziçi University

1998

DEVELOPMENT OF A VACUUM CLEANER MOBILE ROBOT

APPROVED BY:

Doç.Dr.H.Levent Akın .....
(Thesis Supervisor)

Doç. Dr. Feza Kerestecioğlu .....

Doç. Dr. Yağmur Denizhan .....

DATE OF APPROVAL : 22.06.1998

ACKNOWLEDGMENTS

I would like to thank to my thesis supervisor, Assoc. Prof. Levent Akın, for his patient guidance, encouragement and understanding throughout my study. His experiences and ideas were very helpful.

I would like to thank to Mühendisler Elektronik A.Ş. for supplying the electronic equipments needed.

Special thanks to my family for their understanding and help throughout of this study.

Alp Sardağ

ABSTRACT

This study presents the results of the development of an autonomous mobile robot which is a domestic autonomous vacuum cleaner. The robot is designed according to some new concepts established in this field during the last decade. These principles are artificial life, subsumption architecture and other bottom-up methodologies. These ideas have been applied to the complete robot design, spanning from the shape of the robot to the sensors, from the electronics to the software control structure.

We have built the robot and programmed it to perform various reactive behaviours. The autonomous vacuum cleaner performs area coverage in a room by combining distinct behaviors. The robot has eleven sensors which are used in the algorithms examined. The robot behaviors allow it to avoid obstacles, follow walls, break out of confined areas, detect floor and find a docking area. The performance tests are done through a dedicated simulator and by observation. The experiments performed here were both in the simulator and real-time it appears that the best area coverage algorithm among those tried consisted of a random walk with a probability of 0.10 for following a wall after each encounter with an obstacle.

ÖZET

Bu çalışma kendi kendine hareket edebilme yeteneğine sahip bir elektrikli süpürge robotunun gelişimi üzerinedir. Bu robot son yıllarda oluşan yeni kavramlar üzerine tasarlanmıştır. Bu kavramlar yapay us, dışlama mimarisi ve diğer baştan aşağı yöntemlerden oluşur. Bütün bu fikirler hareketli bir robot tasarımı için kullanılır: Robotun şeklinden algılayıcılarına, elektroniğinden kontrol yazılımına kadar her kısmı bu fikirler tarafından yönlendirilirler.

Robotu tasarlayıp, meydana getirdikten sonra bir takım tepkisel davranışlar göstermesi için programladık. Özerk çalışabilen bu hareketli elektrik süpürgesi robot, kapalı bir alanı bu tepkisel davranışların yardımı ile tarayıp süpürebilmektedir. Robot üzerinde çalıştırılan algoritmalarda 11 adet algılayıcı kullanılmıştır. Tepkisel davranışları robota engellerden kaçınabilme, duvar takip edebilme, kapalı ve dar alanlardan çıkabilme, yer algılama ve park edebilme gibi yetenekler kazandırmıştır. Robotun performans testleri öncelikle yazılan bir simülatör üzerinde daha sonra da robotun gerçek bir ortamda çalıştırılması ile sağlanmıştır. Denenen algoritmalar ile yapılan tüm testler sonucunda en iyi performansın yüzde 10 olasılıklı duvar takibi yapan rasgele süpürme hareketinde sağlandığını ortaya koymuştur.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Page

LIST OF SYMBOLS

a	Acceleration
A	Ampere
Ah	Ampere-hour
C	Coulomb
cm	Centimeter
CPU	Central Processing Unit
DC	Direct current
F	Force
F_1	Force applied by left motor
F_2	Force applied by right motor
F_{app}	Applied force
F_f	Frictional force
F_N	Normal force
F_{net}	Net force applied to the robot
F_w	Gravitational force
g	Gravitational force per unit mass
h	Hour
I	Direct current
I/O	Input output
IR	Infrared sensor
J	Joule
kg	Kilogram
kHz	Kilohertz
L293D	H-Bridge circuitry
LCD	Lithium diode display
m	Mass
Min	Minute
P_m	Power required from motors
RAM	Random access memory
sec	Second
SPI	Serial peripheral interface

T	Torque applied by each motor
T_1	Torque applied by the right motor
T_2	Torque applied by the left motor
V	Volt
W	Watt
W_{in}	Motor's angular speed
W_{out}	Wheel's angular speed
μ	Coefficient of friction
θ	Ramp angle
τ	Net torque applied to the robot

1. INTRODUCTION

Today the mobile robotics field receives great attention. There is a wide range of industrial applications of autonomous mobile robots, including robots for automatic floor cleaning in buildings and factories, for mobile surveillance systems, for transporting parts in factories without the need for fixed installations, and for fruit collection and harvesting. These mobile robot applications are beyond the reach of current technology and show the inadequacy of traditional design methodologies. Several new control approaches have been attempted to improve robot interaction with the real world aimed at the autonomous achievement of tasks. An example is the subsumption architecture proposed by Brooks [1] which supports parallel processing and is modular as well as robust. This approach is one of the first solutions systematically implemented on real world robots with success. Other researchers propose new computational approaches like fuzzy logic and artificial neural networks. Some of them are described by Nelimzov and Smithors in [2].

The interest in mobile robots is not only directed towards industrial applications. Several biologists and psychologists are interested in using mobile robots to validate control structures observed in biological world, some of which are introduced in [3].

All these activities are based on mobile robot experimentation. A simpler way to validate control algorithms is to use simulations, but the simplifications involved are too important for the results to be conclusive. The control algorithm embedded in the robot must consider its morphology and the properties of the environment in which it operates. Real world features and anomalies are complex and difficult to modelise, implying that the experimentation of control algorithms through simulation can only be used in preliminary study but can not prove the success of the control algorithm in the real world. The sole way to validate an algorithm to deal with these problems is to test it on a real robot.

Many robots have been designed to perform experiments on control algorithms but only a few make cost efficient experiments possible. Brooks [4] has designed several robots with effective electronics and mechanics. The control algorithms are programmed in the subsumption behavioural language, taking into account software modularity and real-time and parallel processing. Unfortunately, during experiments, only a few tools are

available to improve the understanding of the control process. Moreover, the custom programming language makes code portability and algorithm diffusion difficult.

The lack of a good experimentation on mobile robots for single-robot experiments, means that it is impossible today to perform collective-behaviour experiments. The programming environment and the real time visualisation tools are totally insufficient for this purpose.

The development of our robot, ROBONE, adresses the problems mentioned above. It is designed to sweep an indoor environment and its performance is observed. Prior to any observation, a tiny robot simulator RoboSIM is written and the performance metrics are derived by simulation. Its reprogrammability causes the robot to be used for other experimental purposes.

The rest of the document will consist of four parts. Part two, backgroud, will give background information on mobile robotics and work done so far on vacuum cleaner mobile robots. Part three, design of an autonomous vacuum cleaner, will give the details of our design considerations and problems encountered. Part four, will deal with performance tests done on the implemented mobile vacuum cleaner robot. The final part, conclusion, will summarise the whole work done and give some idea on future work.

2. BACKGROUND

Building a mobile robot covers different aspects of engineering: Electrical engineering, software engineering and mechanical engineering are all involved in building a mobile platform. In this chapter, a survey about previous implementations and relevant issues will be given.

2.1. Mobile Robot

Robotics is about building systems: Locomotion actuators, manipulators, control systems, sensor suites, efficient power supplies, well-engineered software, all of these subsystems have to be designed to fit together into an appropriate package suitable for carrying out the robot's task. Performance, accuracy, robustness are only some of the design considerations in robotics.

Despite the unsolved problems, there arised the need of using mobile robots. What makes a robot mobile is its ability to move with or without human assistance. If there is no need for human assistance then the robot becomes autonomous. Designing autonomous mobile robots that are suitable for real world conditions is the hardest job in the robotics arena because of chaotic dynamics of real life.

Being autonomous, is the ability of making decisions. The decision of the robot highly depends on the robot's real world representation and its level of information about its environment. The first factor can be tuned by software, however the second factor highly depends on its sensor suite. Sensors are the only way an autonomous mobile robot can get information about its environment.

Sensors vary according to their capabilities, price and quality [5]. Some of the commonly used sensor types are: Tactiles, that can sense a collision or act as a DIP switch; infrared sensors, that sense an obstacle at a certain distance off its attachment point; photocells that sense light; sonar sensors, that sense the distance till next obstacle; compasses that senses the magnetic direction of the robot and so on so forth.

The mechanical and electrical base of the robot altogether is an another important factor. The choice made at this step may determine the speed-torque trade off, the rotation capability, etc. For example, if the robot will be an all-terrain one, like NASA's Pathfinder [6] robot used in Mars Exploration Project, it may be desired to be as stable as possible, since it may come against a small rock or a small hill or even a crack in the ground filled with some brittle substance, such as ice. Thinking the money invested in Pathfinder, it should explore Mars's surface with minimum harm and maximum information about its environment. Likewise, a robot that will be used in a house needs to be more robust since its environment is dynamic. Humans have a bird's eye view over the furniture while a robot acquires them as walls, thus interprets the house as a maze. Even a human may lose his/her way in an unknown city. However humans learn! If such an explorative function can be embedded to a mobile robot, theoretically, it will give better and better performance as it acquires knowledge. Humans also have long lifetimes. They can feed to live, sleep to refresh. An autonomous mobile robot must have such capabilities. It must be able to recharge itself or else stop and indicate to a human that it needs recharging where the autonomous action is wounded.

Today, mobile robots are used for several purposes. They are used for pool cleaning; as guards where they can sense fire, dangereous gas; for space applications (i.e. Pathfinder); as porters [7], etc. Autonomous action in household jobs such as cleaning, sweeping, cooking and such is not yet achieved.

2.2. Control Loop

The most obvious way of controlling a robot is to give it tasks to do, without any concern about the environment around it. This form of control is called open loop and consists of a simple signal being given and an action being carried out. The robot does not make any sort of check as to whether the correct action was completed, rather it mechanically follows the steps in a prescribed pattern. Closed loop control involves giving the robot feedback on the task as it progresses to allow it to ascertain whether the task is actually being completed.

Most industrial robots support minimal handling of unpredictable events: the tasks are fully predefined (e.g. welding certain automobile parts together) and the robot has no responsibility with respect to its environment (it is rather the environment that is responsible for not interfering with the robot.) The open control loop architecture applies to this situation. The robot initiates an action or series of actions without bothering to check on their consequences.

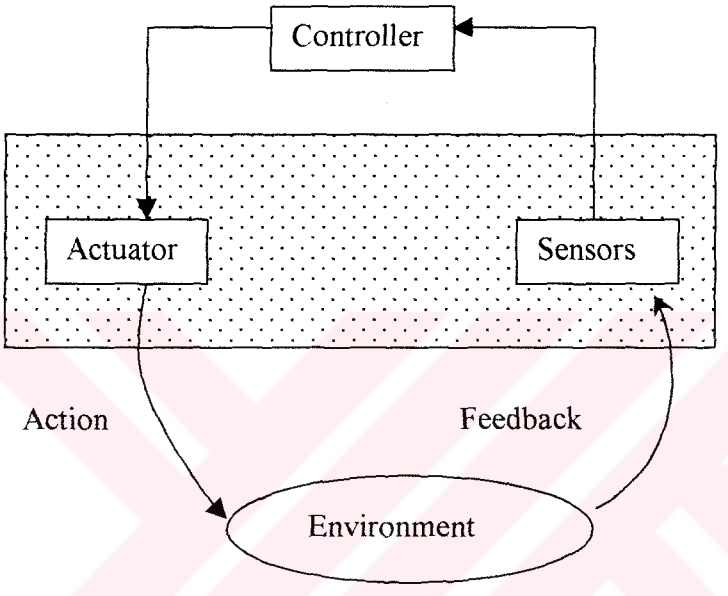


FIGURE 2.1.The closed loop control architecture.

Upgrading this architecture to mobile robots involves adding feedback, thus producing a closed loop architecture. The controller initiates robot actions and monitors their consequences, adjusting the future plans based on this return information. The Figure 2.1 shows the model of the closed loop control architecture.

An advantage of the closed loop architecture is the simplicity. It captures the basic interaction between the robot and the outside. The vacuum cleaner robot has to perform its actions in a real time and dynamic environment. Therefore a control loop architecture is a must for such robots.

2.3. Approaches in Mobile Robot Software

In mobile robot programming, there are two approaches:

- Traditional Approach
- Subsumption Approach

2.3.1. Traditional Approach

The traditional approach [8] is based on the ideas of world modelling and planning. This approach decomposes a robot program into an ordered sequence of functional components as shown in Figure 2.2.

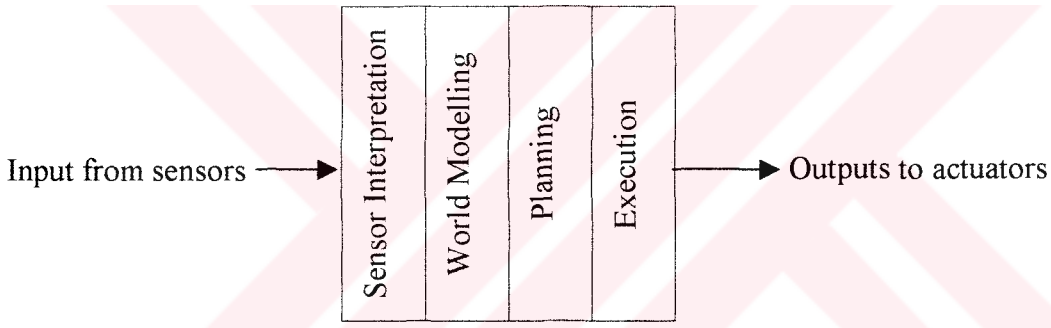


FIGURE 2.2. Functional components of traditional approach.

First, data are collected from all sensors. Noise and conflicts in data are resolved in such a manner that a consistent model of the world can be constructed. The world model must include the geometric details of all objects in the robot's world and their positions and orientations. Given a goal, usually provided by the programmer, the robot uses its model of the world to plan a series of actions that will achieve the goal. Finally the plan formulated is executed by sending appropriate commands to the actuator. An example is HANDY [8], the world-model approach with a brief example using a modelling/planning system called HANDY, which was recently completed at the MIT Artificial Intelligence Laboratory. HANDY is a task level planning system for manipulator type robots that can solve the pick-and-place problem. The pick-and-place problem takes as inputs a model of the world, a part at some location and orientation, and the desired final location and orientation for that part. The pick-and-place problem then is solved if the program can compute a detailed

set of robot motions (and gripper openings and closings) that will move the part from its origin to its destination. Thus, the robot should be able to pick up the correct part from its origin to its destination. In some assembly tasks, the real robot system first uses a laser scanner to identify the position and orientation of the part to be moved. It then incorporates this information into a geometric model of the world, provided by the programmer. The programmer also specifies the desired final position and orientation of the part. HANDY then uses several sophisticated planners, first to plan how to grasp the part and then to plan gross motions of the robot arm to move it into the vicinity close to the part. Finally, after all plans have been formulated and all constraints satisfied, HANDY executes the plan by sending a long series of commands to the robot. These commands specify precisely each small motion the robot must make and when to open and close the gripper. This modelling/planning approach has strong appeal, principally because of guarantees and optimisations makes it possible. There are planning strategies that, in a finite amount of time, will compute a sequence of motions guaranteed to accomplish the task or prove that the proposed task is impossible. Such guarantees would also appeal in the mobile robot domain. For instance, a mobile robot that used such global information about its world to formulate a plan would never fall into the trap of following a path to a dead end and having to backtrack. Instead, it would always choose the most direct route from start to goal. Unfortunately, the modelling/planning approach has some disadvantages that make it impossible to use in an application such as a vacuum cleaner robot.

The drawback of the world-model schemes is that they require large amounts of data storage and intense computation. This drawback is not necessarily a concern for a manipulator type robot, but it can be for a mobile robot, which must carry its computational resources on its back. The HANDY program is composed of more than 100 high level Lisp files and requires a powerful computer with several megabytes of RAM to perform satisfactorily. Because the natural world is enormously rich in detail, schemes to represent it simply require a large number of bits.

Many of the advantages of the modelling/planning approach come from its ability to use global information. A program that takes into account all relevant information can be expected to produce better result than one that makes all decisions based on local (i.e., only some) information. It is the internal representation of the world that makes possible

the use of global information, but problems occur in the construction of this model. For a plan to be reliable, the model on which it is based must be highly accurate. This requires high-precision sensors and careful calibration, both of which are costly. Even the best available sensors suffer from several difficulties: Sensor data are unavoidably noisy. Sensors are subject to systematic errors, and different sensor technologies often produce conflicting results when measuring the same quantity. For example, sonar and infrared ranging systems may give different distance readings because of the surface properties of the objects at which they are aimed. Typically a modelling/planning must devote considerable resources to figuring out the most likely interpretation when presented with inconsistent data from a single sensor and conflicting data from multiple sensors into one data structure, the world model. This process is known as sensor fusion. Some programs rely on the programmer rather than sensors for building most or the entire world model. Synthesising a world model can reduce the burden of interpreting sensor data, but unfortunately doing so can also limit the robot's ability to respond autonomously to changes in its environment.

The modelling/planning paradigm is by nature sequential. The approach first takes the snapshot of the world, then processes the acquired information and the acts. If the world happens to change between the snapshot and the action, the plan may fail. Trying to make actions of such a program more intelligent may produce undesirable results. The more time the program devotes to resolving conflicting sensor data, to refining its model of the world and to optimising its plan, the longer the delay between sensing and acting. This delay increases the chance that a significant change will occur in the world, thus invalidating the plan.

2.3.2. Subsumption Approach

Rodney Brooks at MIT Mobile Robot Lab proposed the subsumption architecture [1] that is a way of organising the intelligence system by means of layering task-achieving behaviours without recourse to world models or sensor fusion. The word subsumption is used to describe the mechanism of arbitration between the layers of task achieving behaviours. Arbitration is the process of deciding which behaviour should take precedence when much conflicting behaviour are triggered. In a subsumption architecture, the designer

of the intelligence system lays out the behaviours in such a way that higher-level of behaviours subsume lower-level behaviours when the higher level behaviours are triggered. The main idea is there are no explicit geometric representations of the world from which the robot plans its actions. Instead, there are a number of control loops granting a very tight coupling of perception to action and from the interaction of much simple behaviour, complex activity seems to emerge.

The subsumption approach, a promising new alternative to the modelling/planning paradigm has recently proposed by Brooks at the MIT. Brooks' subsumption architecture provides a way of combining distributed real-time control with sensor-triggered behaviours. Subsumption architecture, instead of making explicit judgements about sensor validity, uses a strategy in which sensors are dealt with only implicitly in that they initiate behaviours. Behaviours are simply layers of control systems that all run in parallel whenever appropriate sensors fire. The problem of conflicting sensor is handed off to the problem of conflicting behaviours. Fusion consequently is performed at the output of behaviours (behaviour fusion) rather than the output of sensors. A prioritised arbitration scheme is used to resolve the dominant behaviour for a given scenario. Note that nowhere in this scheme is there a notion of one behaviour calling another behaviour as a subroutine. Instead all behaviours actually run in parallel, but higher level behaviours have the power to temporarily suppress lower-level behaviours. When the lower level behaviours are no longer triggered by a given sensor condition, however, they cease suppressing the lower-level behaviours and the lower-level behaviours resume control. Thus, the architecture is inherently parallel and sensors interject themselves throughout all layers of behaviours. There is no unified data structure or geometric world model.

2.4. Unsolved Problems

Building a robot involves many issues. The robot builders have to deal with bias in the circuits, bugs in the code, slip in the wheels, noise in the sensors and transient in the power supplies. For the process of building a robot, the robot builders need a strong base in the following areas: electrical engineering, mechanical engineering, computer science and artificial intelligence.

Today's robots are not as talented as the ones in science fiction movies. The gap between expectations and experiences for the beginning robot builder can be daunting. The crux of the problem is that humans are very capable. They take many things for granted in their biological selves: the acuity of the eyesight, the power-to-weight ratio of the muscles and efficiency or the energy conversion system. In fact, the disparity between expectations and experiences grows even wider in the comparison between the tiniest insects and today's robot. Even their perceptual-motor skills are amazing. Common houseflies can land upside down on ceilings, spiders can assemble most intricate homes and ants can carry loads many times their weight.

2.4.1. Navigation

There are many unsolved problems in mobile robotics. One open question has to do with what is involved in endowing a robot with the ability to navigate its environment. Salmon can locate their spawning grounds from thousands miles away, pigeons can find their destinations on either sunny or cloudy days and bees can make a beeline to a food source in a quick response to another bee's dance. By contrast, few robots can make it down the hallway without recourse to humans modifying the environment with artificial landmarks.

The underlying issue here is one of representation. What sort of computational structures are required to grant competence in navigation? How far can reactive systems be stretched? Are world models and sensor fusion required at some point?

2.4.2. Recognition

Another problem that goes along with navigation is recognition. Landmark recognition in a generally unstructured environment is a very hard problem. Whether using cameras, pyroelectric sensors, force sensors or microphones, recognising patterns in the environment is not trivial. Recognition problems can be computationally intensive and subject to complexities because of lighting, occlusion and noisy data. It is usually hard task to differentiate furniture gaps between open doors and decide whether to move in or not.

2.4.3. Learning

As the mobile robots become more and more complex and as we attempt to make them more sophisticated, perhaps by incorporating more sensors for greater perceptual acuity or by adding more actuators for finer dexterity, the software can become severely strained in trying to deal with so many inputs and outputs. One area of research is to investigate how, when and where learning algorithms can be incorporated into a robot's intelligence system to alleviate the programmer's burden. What are the right types of things to learn? Can a robot learn to calibrate its sensors? Can a mobile robot learn new and better behaviours?

2.4.4. Thoughts

Twenty years ago, nobody would have believed a computer would be used as a fuel-injection controller in every car. Back then, when computers filled entire rooms the people who built the first microprocessor could not imagine their invention supported by some memory and a control logic would change the way the world works.

Likewise, tiny single board robots consist of a controller, sensors and battery. However, many of these small robots may just change the way human beings' think about solving problems.

2.5. Mobile Robots in Vacuum Cleaning

The domestic autonomous vacuum cleaner that cleans a house while the owner is outdoors doing some shopping, is an old wish and it may stay so for some time. Many companies and research institutions have tried to develop such a robot, with only few results, some of the trials are presented in [9]. The human operator can not be replaced easily by artificial intelligence. The lack of results is common to a large range of applications in autonomous mobile robots and it is because of the complexity of the task and the limitations of the actual technology. Autonomous, in this case, does not simply mean that the robot has batteries, good computational power and good behavioural rules, but much more. As Steels explains very well in [3], autonomy is not simply composed by a

set of smart rules, but is the ability to create its own rules. To move around, finding a path or doing a job like cleaning seems very simple for us and for many animals. The difference between robots and animals is that animals use their brain to solve a problem and we start to understand the complexity of these "simple" tasks when we analyse the complexity of the brain.

In the last decade, new approaches have been tried to bring up artificial intelligence and the computer science community back to the real world to get better results in the field of autonomous mobile robots. This field needs smart systems that are well adapted to their environment through their shapes, sensors, actuators and behaviours. The embodiment of the system is important and brings a new and complex dimension to the problem of artificial intelligence.

2.5.1. Application Specific Mobile Robot

A new approach must be followed to design an autonomous vacuum cleaner working in an unknown and unstructured environment. As it is necessary to develop the behaviour of the robot, it is also necessary to optimise its shape and its sensor system. All these three criteria (shape, sensors and software) must not only be designed individually but as a whole. Only with a total integration of hardware and software can make such an application become feasible. This integrated design is called an "Application Specific Mobile Robot" (ASMR.)

Developing proper shape is essential for the success of such an application, it will be more compatible with the sensor system and facilitate the programming of the robot's behaviour. Moreover, only some shapes may be appropriate to perform some of the assigned tasks.

The sensor system is an important part of the robot as it provides information about the environment. If the robot is to be used in an unpredictable environment autonomously, it can only rely on the sensor data. The sensor suit mainly defines the environment. The better adapted and the more reliable this system is, the better the programming of the robot will be.

Finally the control system is responsible for the "intelligent behaviour" of the robot. Benefiting from a well-designed robot, the program will be shorter, faster, less complicated and more reliable.

2.5.2. Autonomous Vacuum Cleaner

The commercial autonomous vacuum cleaner [10] can vacuum horizontal and open surfaces. The sensory arsenal employs a floor-level optical input vision system to sense walls, doorways and such, all of which have "vertical" signatures. A multiplex of sonar sensors are used to follow walls at distances up to 30 feet with high precision and help in measuring distances of unpredicted obstacles. Staircase detectors stop the machine should it, in a misguided cleaning fervour, approach a stairwell. For the safety of humans who may be present while the robot is cleaning the entire base of the machine has a contact strip commanding emergency shutdown.

Another control subset is called "Hansel and Gretel." In the well-known fairy tale the innocent tots enter the forest leaving a trail of breadcrumbs to guide them on their return journey. Of course, birds eat the breadcrumbs and the guiding path is lost. In a modern-day analogy, the cleaning robot can trace a path around a large, open area, perhaps using sonar following. As it makes the first round it can release some inexpensive artifact (equivalent to bread crumbs) which provides a guide path, one pass nearer the centre of the area for the second circuit of the area. As the robot follows the path its scrubbing brushes clean up the trace just as the birds did to dismay of Hansel and Gretel; but while obliterating one guide path the robot is also laying a new one.

Figure 2.3 shows how areas of various shapes can be cleaned automatically using the "Hansel and Gretel" concept. Once the periphery has been traced the cleaning pattern spirals inwards towards the centre until the entire area is cleaned. There is no need for laborious "teach-repeat" programming and the sensory system need not keep a precise knowledge of its current location in order to be sure of covering the area without missing portions.

Another navigation concept being explored in this system is akin to a sailor's celestial navigation. One or more lights, mounted to a known height above floor level, are acquired by a vehicle-mounted optical system that computes location from measured elevations and azimuth. If only one "celestial body" is in sight, the navigator's concept of a "running fix" can be employed along with continuous dead reckoning.

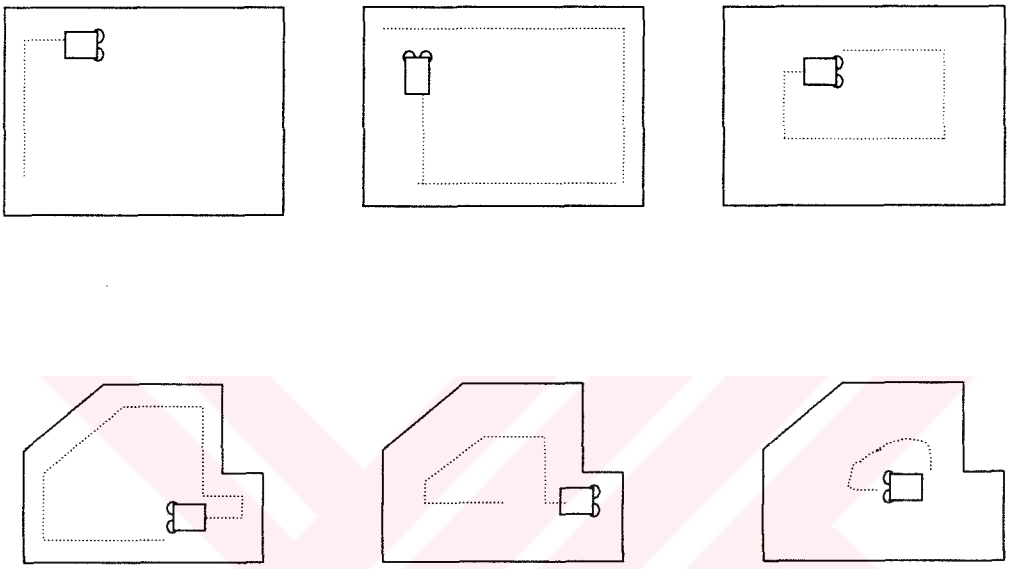


FIGURE 2.3. "Hansel and Gretel" cleaning path control.

2.5.3. Vacuum Cleaning Project at Swiss Federal Institute of Technology

The vacuum cleaning agent includes 16 IR's, 54 Tactiles, a compass, two shaft encoders [8]. The controller includes four microprocessors working in parallel. The main processor is a Motorola 68331, with one megabyte of RAM, 256 kilobytes of flash RAM and 128 kilobytes of ROM. The mobile robot is equipped with a two degrees of freedom arm. With the information of the arm, the robot can construct a local map of the obstacle relative to a fixed point, the position of the robot. An interesting analogy exists between this mobile robot and a blind person. If a blind person enters an unknown room, he uses his arm or his cane but not his body to explore the environment. The uses of his arms allow him to keep his body as a point of reference during the exploration. Moreover, his hands have a large density of tactile sensors, which allow him to identify the objects.

The robot is able to create map of the peripheral of the room by following walls using compass (but the magnetic flux in an apartment is not constant and is highly influenced by metallic objects and electronic equipment) and odometry but not the interior of the room. One way would be to use the created map of the border and to find the shortest path to cover the whole surface by using well-known AI techniques. But this approach would not work well, as the interior of the area is not known. There will likely be some obstacles in the middle of the room that have not been detected during the exploration of the border. Hence this so-called the optimal path would have to be recalculated each time the robot comes across a new obstacle. In addition, as the first generated path is based on incomplete information, the final path will be far from optimal in many cases. It may become stuck at complex environments. In addition, the robot can only clean an area of two or three metersquares.

3. DESIGN OF AN AUTONOMOUS VACUUM CLEANER ROBOT

3.1. Introduction

This chapter outlines the idea behind the work done on the development of mobile vacuum cleaner agent prototype. The robot is expected to sustain basic vacuuming and control functions such as object avoidance, floor detection, wall following, seek open areas and break out of confined areas. Construction of an autonomous vacuuming tool offers a challenging engineering task, but lack of a satisfactory area coverage algorithm will render such a tool ineffective. The construction of the robot consists mainly of four parts :

1. Hardware: The control circuitry, motor interface, vacuum circuitry, sensors.
2. Software: Obstacle avoidance, floor detection and area coverage algorithm that the prototype carries on.
3. Mechanical Assembly: The mobile platform the robot resides on.
4. Tests : Performance evaluation by writing a simulator for the robot.

3.2. Design Goals and Decisions

As a starting point, the basic tasks the sweeping agent is to perform, were defined. It should :

- Move: This part is actually the first goal and effects the whole design. The robot should continue its operation in an unknown environment and should not stuck to an obstacle, should break out of confined areas, turn back or left, right, move back or forth. A sensor driven control mechanism would be quite adequate for these purposes.

- Cover the area: This is a crucial point in the design. All constructions (mechanical and electronical) and implementations (basic movement algorithms) done so far help to implement the best area coverage algorithm implemented on this robot.

In order to accomplish above tasks, the basic control algorithms should be implemented:

1. Obstacle avoidance
2. Floor detection
3. Collision detection
4. Wall following
5. Claustrophobia

In all the points mentioned above one major objective that was always in mind: "it should be cheap enough after the mass production for everybody who doesn't like or have much time to sweep and also would have a satisfactory performance" (one of the main engineering trade offs to be kept in mind.)

In order to achieve the goals described above, the design problems and their solutions were categorized into following subsections:

1. Steering
2. Vacuuming Platform
3. Robot Shape
4. Motor Selection
5. Sensors and their brands
6. Power Supply
7. Control Circuitry
8. Glue Selection
9. Software Development

3.3. Steering

The robot's steering constitutes the major design problem. It effects the rest of the following steps. Changing the steering means changing everything from mechanical parts to software parts. The alternatives were: Synchro Drive, Tricycle Car Drive, Ackermann Drive and Differential Drive.

Considering the mechanical complexity and robot's functionality the following steering types were taken into consideration.

3.3.1. Single Motor Drive

In this design, there is a semi-dependent castor in the front and two wheels at the back equivalently driven via one motor. This was the simplest steering technique in hand so this technique was adopted in the very beginning.

This steering works as follows: If the motor is powered in forward direction, the robot moves straight ahead, when it is powered in reverse direction the semi-dependent wheel turns 60 degrees right and, thus, the robot turns left. Figure 3.1 shows such a path of the robot. The robot turns its head to left while moving back. The more it moves back, the more it turns.

In order to construct up the mechanical platform, using a toy was a good choice since many mechanical problems are solved already. Initially, using a toy airplane, which exactly satisfies the design mentioned above, was considered. But there were the following problems:

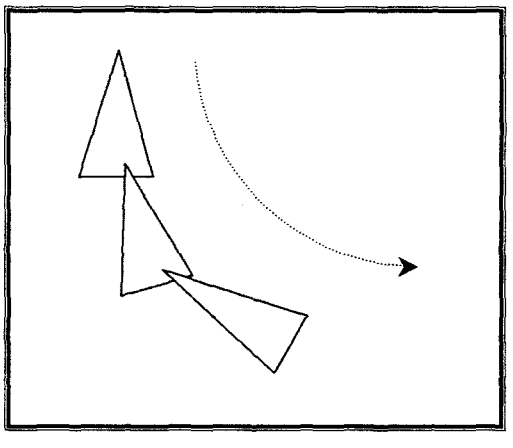


FIGURE 3.1. Track of a tricycle robot.

- The plane (robot) could turn only in one direction and only when the robot was moving backwards. This would make the software development process harder, maybe impossible.

- The toys have motors with 6 V on the average and as the weight of the robot increases these motors can even be destroyed because of the overload. Replacing them with a say 9 V motor was not possible since their gear set was not suitable for motors' gear sets found in the market. They were highly brittle for such motors. Moreover, they could not yield high torque for a heavy design such as a vacuum cleaner.

- Slower movements are required in order to do the sweeping efficiently. For this purpose additional gear sets must be assembled to the system to reduce speed in favor of high torque. This is a really hard task since these toys are marketed in a compact gear box with motor enclosed. External gears could not be added to them.

- As the wheels' radius were small and the motor's power was weak, it was impossible for the robot to surpass obstacles with one or two centimeters height. This is a great problem especially if the robot wants to pass from the floor to the carpet.

3.3.2. Differential Drive Steering

In order to allow the robot to turn in any direction, a differential drive platform is more suitable. Actually a plane goes through three fixed points so a pivot point is needed in such systems. All the problems encountered with the previous choice in mind, it was decided that a paletted platform would solve most of them. The reason behind this choice would be discarding of castors to be used as pivots. Another problem, surpassing obstacles of one or two centimeters height would also be solved by using a paletted platform. Here, a toy tank as the basis was a good choice. In addition to this, the tank was driven by two motors to obtain differential steering, thus having the ability to turn in either direction and around itself. The pallets performed the driving action. The difference between the drive power to two motors would effect the movement of the tank. This design was superior to the first design. In order to turn left in forward direction the process was just to increase the power to the right motor or stop the left motor or just apply reverse power to the left motor. The more the difference was, the smaller circle the robot rotated around. Figure 3.2 shows the movements.

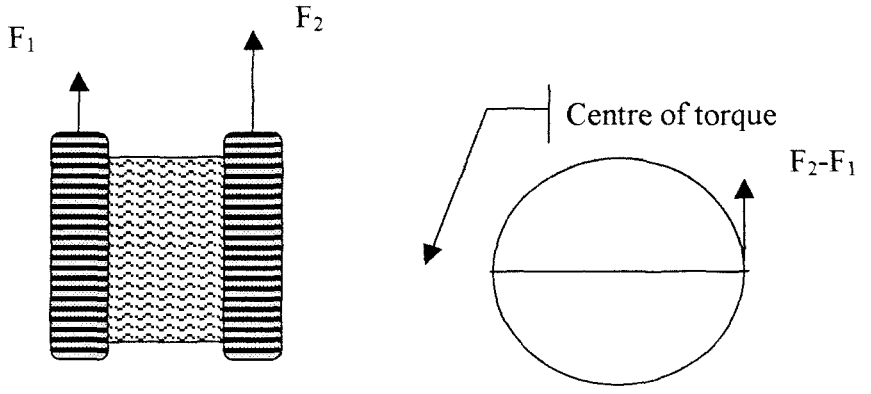


FIGURE 3.2. A big radius rotation.

The difference between two driving forces applies a torque to the retarding side. Actually the centre of the torque should be calculated relative to the centre of mass.

$$\tau = F_{net} * d \qquad (3.1)$$

where d is the distance to the pivot point and F_{net} is the difference of the forces applied to each side indicated by F_1 and F_2.

The most striking turn was obtained when the motors were applied maximum power with opposite directions. Then the robot turned around itself very fast since the centre of torque was located inside the robot in such a circumstance. This was a very useful feature for parking, moving in narrow passages (dead ends) and when where there was not much space to cover. But this much force applied to paletted wheels in an environment with high friction ratio put a great load to the gears. There could be worse situations where the gears would slip on each other causing them to deform or would even break teeths of another and making the robot immobile.

When opposite forces are applied to the robot's pallets, the centre of torque will be the robot's centre of the mass. The more the difference the faster it will turn. Figure 3.3 shows such an instance, where T_1 and T_2 are the torques applied by F_1 and F_2 respectively.

In order to relieve this load on the gears and at the same time to keep turning area at the minimum, it was decided to apply power to one motor while the other was idle.

If the power is applied to the left pallet and the other is stucked the robot turns to right around the other wheel with a speed proportional to the applied power. However, this puts an overload on one of the gears. If the gear set is not a good one they can slipon each other and also will make the robot immobile. Figure 3.4 shows such an instance.

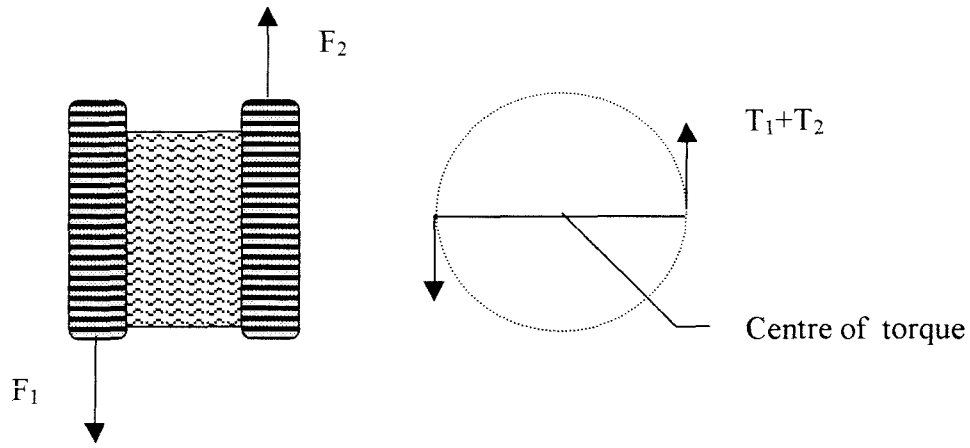


FIGURE 3.3. A small radius rotation.

If the power is applied to the right pallet and the other is stucked the robot turns to left around the other wheel with a speed proportional to the applied power. Figure 3.5 shows such an instance.

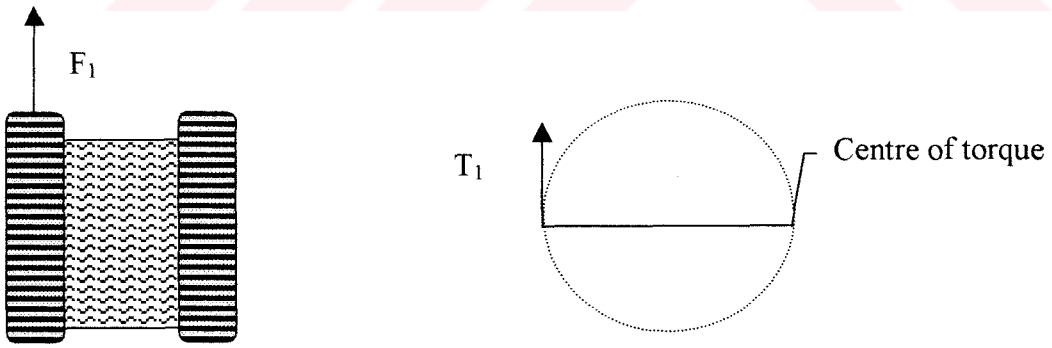


FIGURE 3.4. An average radius rotation to right.

Based on the above discussions, using this platform appeared to be the best choice. The idea was to prefer decreasing mechanical complexity in favor of increasing electronic and software complexity. This is the most reliable and cost effective trade off. The electronical pieces are cheaper than mechanical ones and more abundant. In addition to these Ackerman and Tricycle Drives were not suitable for navigation in narrow spaces.

Synchro Drive was a very good choice but because of its cost and inavailability it was not adopted.

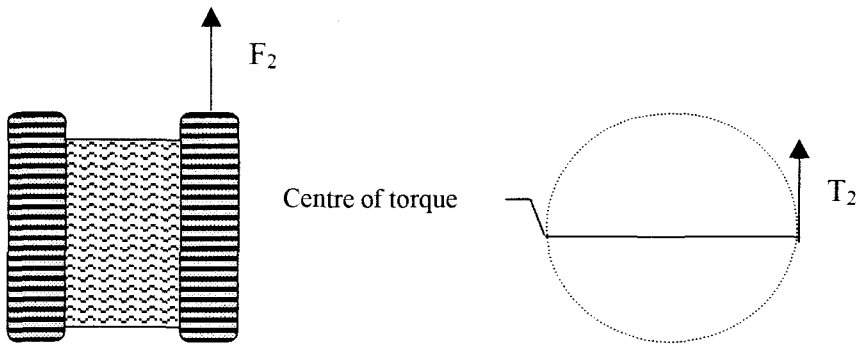


FIGURE 3.5. An average radius rotation to left.

3.4. Vacuuming Platform

For the vacuum cleaning task of the robot, a vacuum tool platform, operation and power source should be obtained. They should be small enough and light enough for the mobile platform to carry them. There were several alternatives: The first one was to use an off-the shelf battery powered hand vacuum cleaner, actually this was an expensive solution but required no mechanical consideration. The second one was to use a fan powered by a direct current (DC) motor.

A CPU cooler fan was used since it was light, cheap and could be found in many sizes and power and a tiny DC motor was already built into it. As experiments were made, they showed that it vacuumed small particles, like dust, but positively it was not suitable for large particles. Assuming this was only a prototype, the fan could also be replaced with a more powerful one, using a motor driving circuitry. The one used was a 12 V and 0.08 A CPU cooler.

The vacuuming platform is placed at the bottom middle. A round piece is taken out of the bottom and the cooler is replaced there. The cooler is patched on the platform by a heater. Plastics easily melt on heat and it is hard to break them apart once patched carefully. We attached a piece of ribbon around the fan to improve the vacuuming power. The appearance of the vacuuming part is shown in Figure 3.6.

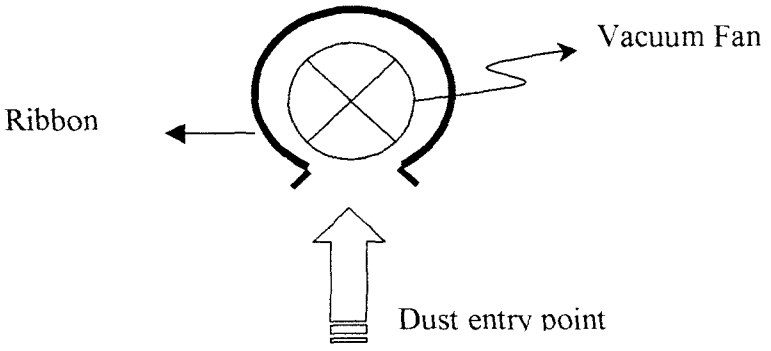


FIGURE 3.6. Vacuuming part.

The actual appearance is also shown in Figure 3.7.

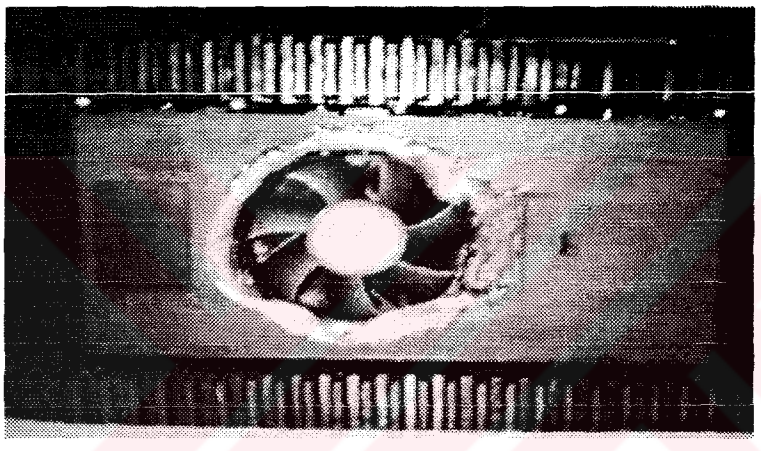


FIGURE 3.7. Picture of the vacuum cleaner part.

For the dust bag a dust cover was needed so that it should not pass dust through, that is it should keep all the dust vacuumed. The best one was to use a piece of mat but since our platform is very compact and sewing was hard with a more practical alternative was required. A thick woven sock was used for this purpose. Then there occurred the need of finding a way to replace the dust bag. When the dust bag becomes full, the user should be able to replace the bag easily. The dustbag is placed inside the tank so we had to design the tanks top cover to be easily removable and in such manner easily replaceable. There occurred some problems. The control and other electrical should be replaced such that removing the top would not require rewiring the circuitry. So the wires were routed from the back to the control circuitry. This way the top could be moved aside, one side attached as a pivot point to the car and making dust bag-changing task easier for everyday use. Such a removal is shown in Figure 3.8 and in Figure 3.18.

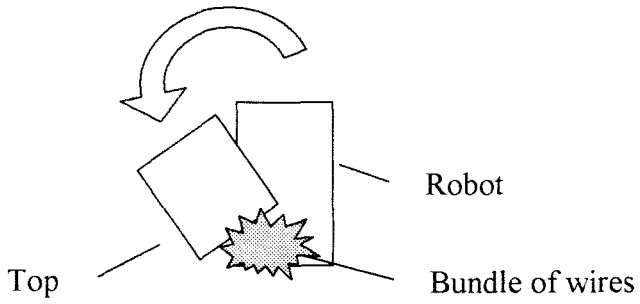


FIGURE 3.8. Removal of robot's top cover for dustbag replacement.

3.5. Robot's Shape

The robot's shape has an important influence on the facility for covering a surface, which is the main task of an autonomous vacuum cleaner. In general the possible solutions can be classified into two categories: rectangle shaped robots and round shaped robots. The round shaped robots have the advantage of facilitating the navigation around obstacles. But round robots can not clean in corners, which is a big disadvantage for potential markets.

An abundance of inexpensive and readily available mobile platforms are adaptable for use as mobile robot basis. As explained before, the robot is mounted on a toy tank. Thus, less design and construction was required. A major problem, component placement (wheel and carouser) was solved by the manufacturer. The appearance from a side is shown in Figure 3.9.

3.6. Motors and Gears

A few years ago, a computer was the largest and most expensive component of a robot, while motors and batteries consumed only small percentage of the budget. These days, while the cost of motors and batteries have changed little, the relationship has changed. For the robot's needs, a DC motor usually runs at a too high speed and at a too low torque. In order to reverse these characteristics, a DC motor must be geared down. Connecting the shaft of a motor to a gear train causes the output shaft from the gear train to rotate much more slowly and to deliver significantly more torque than the input shaft.

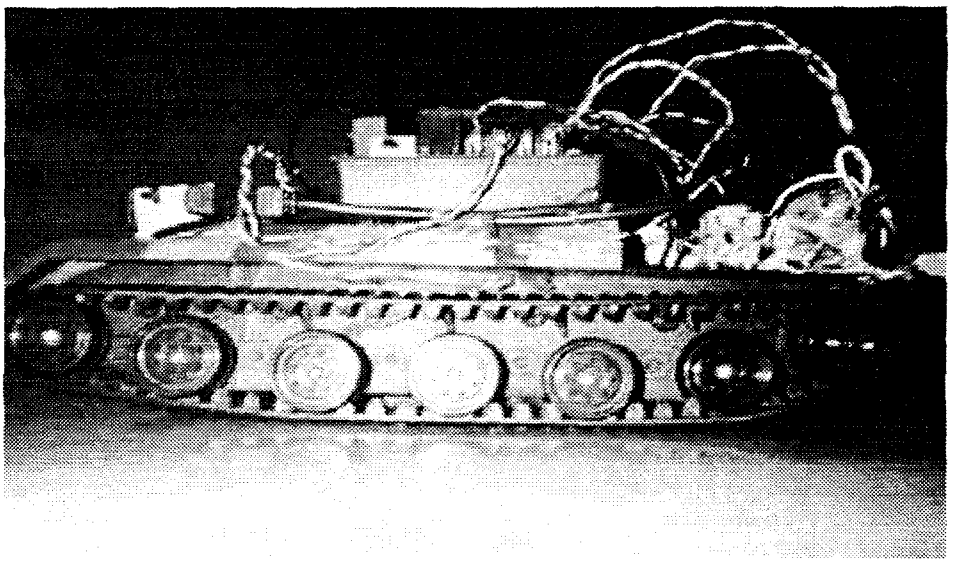


FIGURE 3.9. Side appearance of a differential drive palleted platform.

3.6.1. Gears

Gear trains and transmission systems come in a variety of forms, such as spur gears as shown in Figure 3.10, planetary gears, rack-and-pinion systems, worm-gears, lead screws and belt-and-pulley drives. High-quality gear trains are usually metal, but plastic gears are often found in toys. The DC gear head motor shown below had a spur gear set on the output shaft. Spur gears are most common forms of gears found in DC gear head motors. For the robot we chose this type of gear set. And the final speed is chosen as $W_{in} = 0.001\ W_{out}$ where W_{in} is the motor's angular speed and W_{out} is the wheel's angular speed.

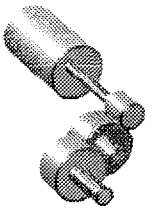


FIGURE 3.10. Spur gears.

3.6.2. Interfacing Motors

A microprocessor cannot drive a motor directly, since it cannot supply enough current. Instead there must be some interface circuitry so that the motor power is supplied from another power source and only control signals are derived from the microprocessor.

The interface circuitry can be implemented in a variety of technologies, such as relays, bipolar transistors and motor driver integrated circuits. In all technologies, however, the basic topology is usually the same. The circuit is known as an H-bridge and merely consists of four switches connected in the topology of an H, where the motor terminals from the crossbar of the H, as shown in Figure 3.11.

In an H-Bridge circuit, the switches are opened and closed in a manner so as to put a voltage of one polarity across the motor for current to flow through it in one direction (setting up magnetic fields and causing it to turn) or a voltage of the opposite direction for reverse rotation. For example, if switches S1 and S4 in Figure 3.11 are closed while switches S2 and S3 are open, current will flow from left to right in the motor. When switches S2 and S3 are closed and switches S1 and S4 are open, current will flow from right to left, reversing the motor. If the terminals are floating, the motor will freewheel and if the terminals are shorted, the motor will brake.

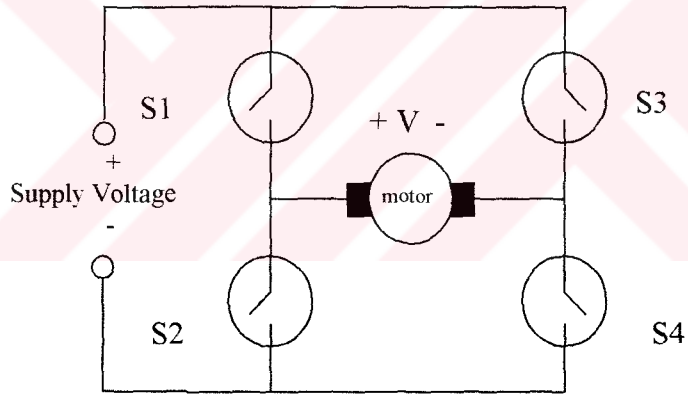


FIGURE 3.11. H-Bridge circuitry.

To control the speed of the motor, the switches are opened and closed at different rates in order to apply different average voltages across the motor. This technique called pulse-width modulation as described in [11]. As mentioned before that the abstractions of switches in Figure 3.11 can be implemented in a number of ways. Relays can be used to turn motors on and off and reverse their directions, but relays are seldom used in pulse-width modulation speed controllers because they typically cannot switch quickly enough and in addition in low voltages (six to 12 V) they cause vibrations. Relays also tend to wear out. Solid-state switches are more convenient for pulse-width modulation schemes

and single-chip solution was adopted in the design, which was already on the controller used.

3.6.3. Power Requirements

In order to estimate the power requirements of the motors for vacuum cleaner robot must be able to deliver, the scenario illustrated in Figure 3.12 was considered. Vacuum cleaner robot uses a differential drive mechanism (two motors) and needs to climb a ramp of angle θ at constant velocity, v. The free-body diagram shown in Figure 3.12 makes explicit the forces acting an the vehicle.

Since the vehicle moves at constant velocity, there must be no net force on the robot. That is, since:

$$F = ma \tag{3.2}$$

where F is the net force, a is the acceleration and m is the mass.

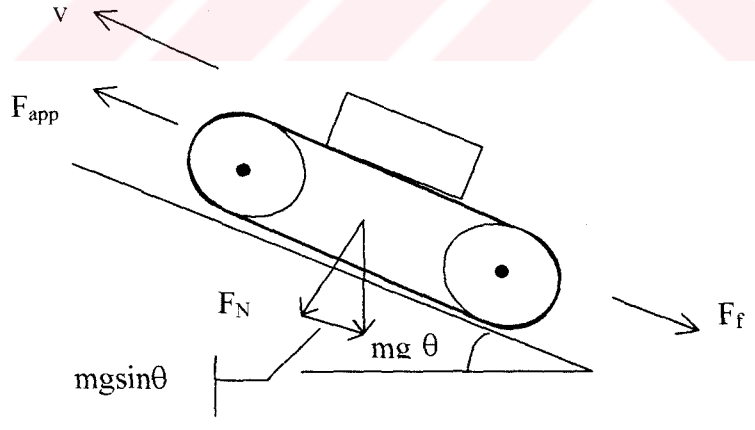


FIGURE 3.12. Free body diagram of the robot on a hill.

When the acceleration, a, is zero (the car moves at constant velocity,) the net force F must be 0. This means that the applied force, F_{app}, from the wheels acting in the direction up the hill must balance the forces down the hill resisting that force. These resisting forces are the friction force and the force that is a component of the vehicle's weight acting in the direction down the hill. Thus:

$$F_{app} = F_f + F_w \tag{3.3}$$

where F_{app} is the applied force, F_f is the frictional force and F_w is the force due gravity.

Here F_f is equal to the coefficient of friction, μ, times the normal force, F_N:

$$F_f = \mu F_N = \mu mg\cos\theta$$
$$F_w = mg\sin\theta \tag{3.4}$$

where mg, mass times the acceleration, is just the weight of the robot. This leaves:

$$F_{app} = \mu mg\cos\theta + mg\sin\theta \tag{3.5}$$

The power required from the motors is the product of the force that needs to be applied by the wheels times the velocity, v, the robot travels up the hill:

$$P_m = F_{app}v \tag{3.6}$$

where P_m is the power required from the motors and v is the velocity.

Each motor is expected to supply half the required power, since the vacuum cleaner has two motors. The model just described above is hardly realistic. Certainly it is not expected that the robot will be climbing up a ramp forever. Rather, since the real world is so complicated (e.g., uneven terrain, stop-and-go crises, unknown coefficient of friction, accident with chair and table legs, etc.) this model is used simply to attempt to size the peak power requirements.

The vacuum cleaner robot weights about 1.100 kg. Assuming that the robot is climbing a 30-degree grade. Its steady velocity is found to be 0.054 m/sec. Picking a value for μ is a way of trying to account for slippage and friction from the treads and the like. It

is not clear what this coefficient of friction will be, but on the average it is 0.4. Then the required power is calculated in equation set 3.7.

$$P_m = F_{app}v = mg(\mu\cos\theta + \sin\theta)v$$

$$P_m = (1.100\text{kg} \times 9.8\ \text{m/sec}^2)(0.4\cos 30^\circ + \sin 30^\circ)(0.054\text{m/sec}) = 0.49\ \text{W} \qquad (3.7)$$

To add a safety factor, both because there are so many unknowns and because the maximum efficiency point is at a much lower torque than the maximum power point, we multiply the power requirement by a safety factor of three, giving:

$$P_m = 1.5\ \text{W or } P_m/2 = 0.75\ \text{W} \qquad (3.8)$$

for each motor.

3.6.4. Motor Selection

At the beginning using the drive and suspension of a toy as the base of a mobile robot seemed to produce a satisfactory result. Less design and construction were required and the problems of mechanical power transmission had largely been solved by the manufacturer. Also, it is often much less expensive to adopt a mass produced toy than to purchase similar component parts separately. But there were the following problems:

- The motors in toys typically require high current and provide low efficiency. For both of the toys used, the motors were six volts, low torque motors. And the current requirement was about 0.125 A .

$$P_m = V \cdot I$$
$$0.75\ \text{W} = 6 \cdot I \qquad (3.9)$$
$$I = 0.125\ \text{A}$$

But what was needed was a high torque (to carry weight) and low current requiring (the control circuitry would supply) motor possibly close to 12 V which will decrease current requirement to half, 0.0625 A. The toy's motors nearly burned the motor drive

(L293D) H-bridge circuitry and making the controller circuit disabled to implement the software downloaded on it. Thus they were useless.

In general, the motors and gearing used by toys are designed to make the toys fast. Thus, control problems because of acceleration and velocity are encountered when the robot is required to move slowly, which is exactly the case here, in order to respond to sensors and to sweep efficiently.

The tank in consideration had two motors. They were both six volts. They transmitted their power to palettes through three sets of gears, so the tank was slowed down and the torque was increased. However, this was not enough. During experiments the control circuitry heated. This was extremely dangerous. The control circuitry could be destroyed. Moreover, the algorithm could not work because of noise induced on the sensors. When the motor driving circuitry got hot, the inputs from sensors were coupled and meaningless data were received from them. At the beginning, using of an external H-bridge circuitry seemed a feasible solution. But after a careful examination this appeared as a cumbersome and needless task. As the engineering principles say, the less complex but successful design should be adopted in a design. We decided on replacing the two motors by 12 V motors. The motors are DC gear head motors and they are very useful in constructing a small robot. DC gear head motors are normally based on permanent magnet ironless rotor motor in order to be as lightweight as possible. On these motors there are also shaft encoders integrally connected. Actually the motors to be used should match the gear set used in the toy. Among several alternatives there was a DC gear head motor with shaft encoders integrally connected as shown in Figure 3.13. They were taken out of the money counting circuitry of an ATM (Automatic Teller Machine.) They yielded the desired performance.

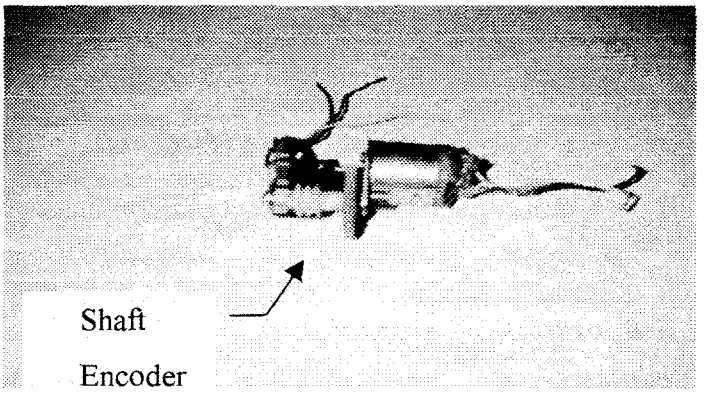


FIGURE 3.13. Picture of a DC motor with a shaft encoder.

3.6.5. Sensors

The sensor suite was chosen according to the algorithms to be used. An appearance of some sensors mounted to the robot's front is shown in Figure 3.14. Since others could program the robot easily (reprogrammability of the robot makes it attractive) we considered the area coverage problem when choosing the sensor suite since navigating autonomously was not satisfactory for a vacuum cleaner robot. We will compare different area coverage algorithms according to their performance namely:

- Random Sweep
- Random Sweep with wall following

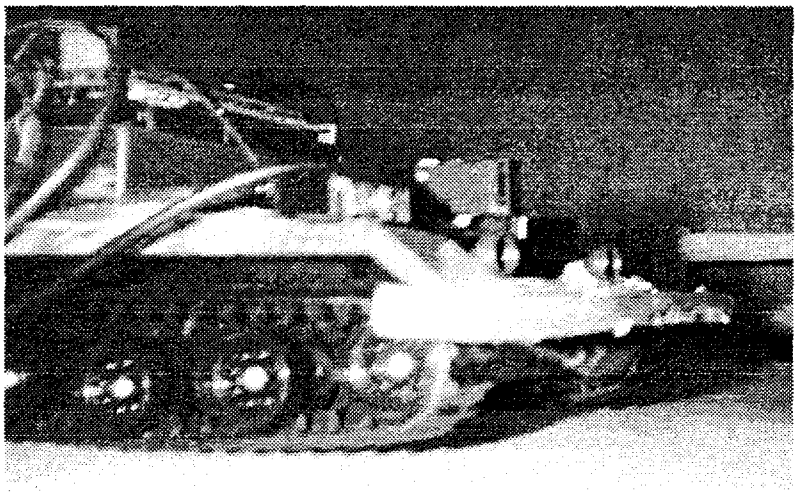


FIGURE 3.14. Front appearance of the robot.

To accomplish these tasks the sensor suit used, their number, their functions and their locations are given in TABLE 3.1.

TABLE 3.1.Sensor suite.

Sensor Type	Function	Location	Number
Infrared (IR)	Obstacle Avoidance	Front	1
IR	Floor Detection	Front Bottom	1
Tactile	Collision Detection	Front	2
Tactile	Collision Detection	Rear	2
IR	Wall Following	Sides	2
Shaft Encoders	Odometry	Back	2
Photocell	Beacon Detection	Front	1

IR sensors vary according to their capabilities. They consist of a transmitter and a receiver. These parts can be either obtained separately or as a whole. If they are separate there are several problems. First of all, they should be calibrated and they should be isolated from external light sources. This is often hard. The whole-in-one types need no such calibration and they are very accurate but are expensive. The ones on the robot are also capable of sensing very dark surfaces as well. The IR sensors can sense dark surfaces hardly, since dark colours absorbs light. The IR sensors are driven via 12V and the power needed is 0.025 A. and when they are used as analog input they give a value in the range of 0-255 and when they are used as digital inputs, they give 0s and 1s. For implementing following behaviours, the strategy is to use near-infrared proximity detectors (20 cm). Although these sensors typically do not return the actual distance to an object, they do indicate whether or not something is present within the cone of detection. The goal of following walls using two detectors is reached as sketched in Figure 3.15. The robot when it sees an object in front, as explained in the code and when there is another object at the right, it decides to turn left until A does not see and B sees the wall, by this way it aligns to the wall and begin to follow it.

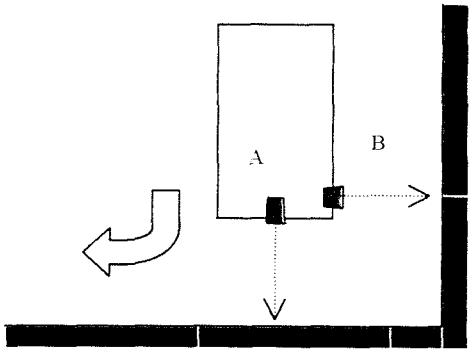


FIGURE 3.15. Corner detection in wall following mode.

Tactile switches are found in the market easily and very cheap. They vary according to their tactile rods. There are long ones and small ones. The vacuum cleaner robot has small ones. Tactiles or Microswitches are momentary switches that are attached to the bumpers to signal when the robot has run into the obstacle as shown in Figure 3.16. They simply ask "Am I touching anything right now?" or "Have I lost contact with something?".

FIGURE 3.16. Two types of microswitches.

The switches are mounted to the bumpers in such a way that, when the robot contacts to an object, one of the switches will close, thus revealing the relative positions of robot and object. There are two bumpers, one on the front and one on the back. They are made of tin, which is suitable for assembly and quite hard enough to accomplish bumper position. They are covered with cable coverings, to reduce the momentum of any bump. The mechanics underlying them are different. The front bumper's mechanics is shown in Figure 3.17.

The rod on the front is the line where any bump occurs. The centre region has no functionality. As the robot bumps into an obstacle or wall, if the bump point is on either left or right region, the bumping force applies a torque to the concerning bumper's tactile. The bumpers act as centres of torque. We needed a centre region to make the bumper more durable and that mostly there are no contacts occur from that region. The bumper at the

back is different from the one in the front in this respect. Since width of the centre region is smaller, the region of sensing is longer.

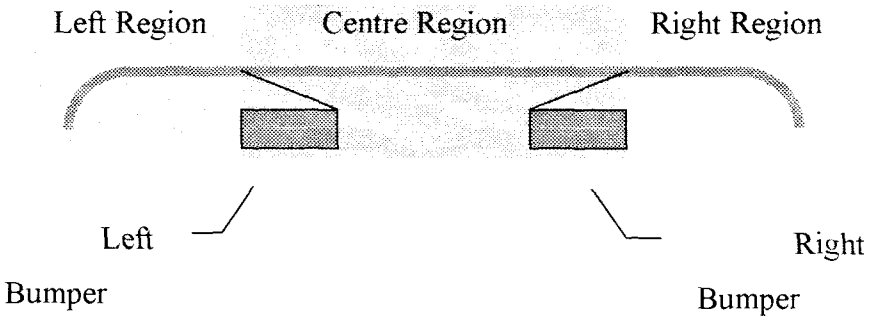


FIGURE 3.17. Front bumper.

Their assembly is very simple. First of all one should note that these have two basic modes.

- Normally Open: This state means the switch should not be pressed normally. If the switch is pressed (if the robot has bumped into an object) then it signals.

- Normally Closed: Actually this state is not used in the robot. However one can use this property to determine if contact is lost with an object.

Shaft encoders are integrally connected to the motor. They are incremental shaft encoders. Each time the shaft turns by a small amount, the state changes from high to low or vice versa. Thus, the rate at which pulses are produced corresponds to the rate at which the shaft turns. But the calculations of odometry calculated based on the information of the shaft encoder are not exact because the robot wheels slip on some surfaces in real world.

Photocells are just like IR sensors. They have two components: Receiver and transmitter. Transmitter is placed somewhere in the room and the receiver is placed on the robot. When the robot wants to stop, it simply moves to docking station mode where it searches for the active beacon mounted on the wall. As soon as it finds it, it moves toward it till it sees an obstacle in the front.

3.7. Power Supply

Autonomous robots must carry their own power source. Solar cells, chemical reactions are some of the ways to derive energy. Batteries are by far the most common solution employed by mobile robots for the solution of energy storage problem. If batteries were not employed then a cable connected to an external power source should have been used, which would form an obstacle to the robot itself. For the vacuum cleaner robot NiCd batteries were used because they were readily available and had low internal resistance as the battery discharged. The first trial was with a one with 12 V 1.1 Ah. It had a high energy capacity, but it was large and heavy and on the robot there was not much space available and its weight required more current output. With this problem in mind, search continued extensively and another one with 12 V 0.8 Ah was replaced with that. The second one was half of the first one both in size and weight and it fitted to the robot as shown in Figure 3.18. The battery housing was enough to keep the battery from vibrations during movement, so it was not attached or glued to the robot.

Typically, battery capacity is not given in joules, but in units of ampere-hours. In order to find energy contained in a battery pack, one must multiply the capacity rating in ampere-hours by the nominal voltage rating of the battery. The battery of vacuum cleaner contains:

$$12\text{V} \times 0.8\text{Ah} \times \text{C/sec/A} \times 3600\ \text{sec/h} = 34{,}560\ \text{J} \tag{3.10}$$

As we mentioned earlier the IR specification used 12 V and 0.025 A. Then the energy consumed in one second:

$$\begin{aligned} P &= VI \\ P &= (12\ \text{V})(0.025\ \text{A}) \\ P &= 0.3\ \text{W} \end{aligned} \tag{3.11}$$

There are four IRs on the robot, the total energy need by IRs in one second is:

$$P = 1.2\ \text{W} \tag{3.12}$$

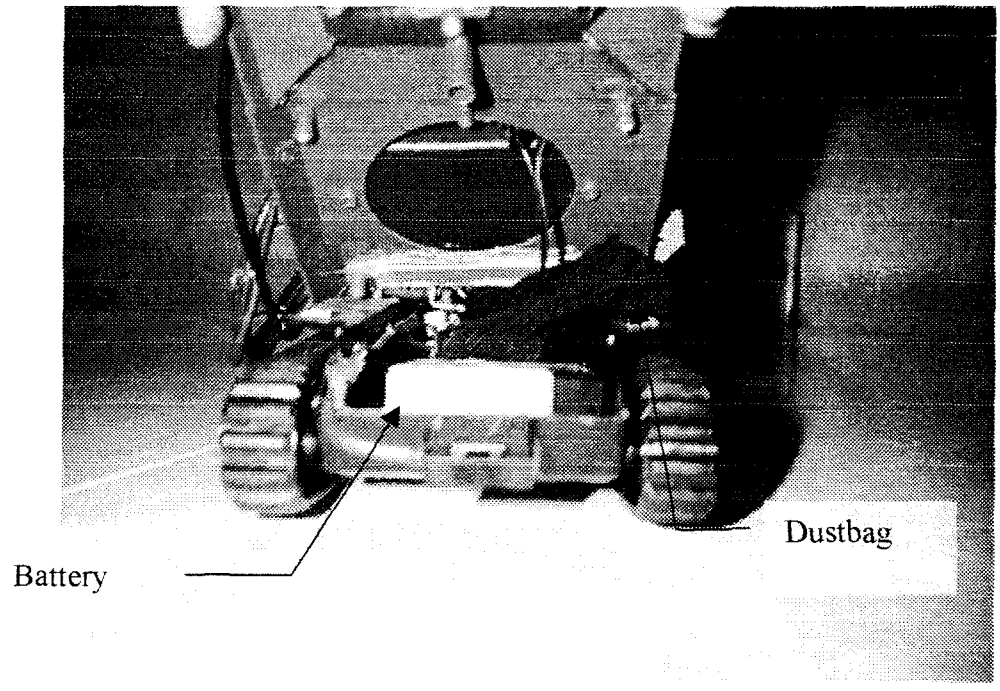


FIGURE 3.18. Battery replaced inside the robot.

For the fan motor the same type of calculation will give its energy need for one second :

$$P = (12\ V)(0.08\ A)$$
$$P = 0.96\ W \qquad (3.13)$$

and the motors and sensor energy needs for one second are given in TABLE 3.2.

TABLE 3.2. Energy requirements of sensors used.

Consumer	Number of Items	Power needs (Watt)
IR	4	1.2
Photocell	1	0.3
Driving motor	2	1.5
Fan Motor	1	0.96
	Total	3.96

Then the battery can support the robot for:

$$34{,}560\ \text{J} = (1.2\ \text{W} + 1.5\ \text{W} + 0.96\ \text{W} + 0.3\ \text{W}) \times (t\ \text{sec}) \quad (3.14)$$

$$t = 8728\ \text{sec} = 2\ \text{h}\ 26\ \text{Min.} \quad (3.15)$$

where t is the maximum time the robot can sweep prior to a recharge.

3.8. Control Circuitry and Hardware

Although analog implementation could be possible, this would provide no flexibility in the design and behaviours. On the other hand, using a microprocessor as the control element would lead to less power consumption, size and ease of use. Using a microprocessor requires programming it. For the control circuitry, Handy Board was used.

Handy Board is a premanufactured control circuitry [8]. The Handy Board features: 52–pin Motorola 6811 microprocessor with system clock at two megahertz; 32K of battery-backed CMOS static RAM; Two L293D chips capable of driving four DC motors; 32 character LCD screen; two user-programmable buttons, one knob and piezo beeper; powered header inputs for seven analog sensors and nine digital sensors; internal 9.6 V NiCd battery with built-in recharging circuit; Hardware 38 kilohertz oscillator and drive transistor for IR output and on-board 38 kilohertz IR receiver; eight-pin powered connector to 6811 SPI circuit (one megabaud serial peripheral interface); Expansion bus with chip selects allows easy expansion using inexpensive digital I/O latches; Board size of 4.25 to 3.15 inches, designed for a commercial, high grade plastic enclosure which holds battery pack beneath the board.

Its controller is the cheapest microcontroller available and it has attractive built-in features such as analog-to-digital converters, system time clock, etc. In other words, it is a good choice to build a single-board robot.

With its peripheral circuitry it is very easy to assemble sensors, motors and others. Its serial connector plug, an interface between the computer and the board, helps one to use

a computer to develop the program and download onto Handy Board. A mini downloader is used for this purpose.

In order to program the robot, an assembly language or a high-level language such as Lisp or C is generally used. Handy Board has a special compiler called Interactive C by Fred Martin [12].

Interactive C (IC) is a C language module consisting of a compiler (with interactive command-line compilation and debugging) and a run-time machine language module. IC implements a subset of C including control structures (for, while, if, else,) local and global variables, arrays, pointers, 16-bit and 32-bit integers and 32-bit floating point numbers.

IC works by compiling into pseudo-code for a custom stack machine, rather than compiling directly into native code for a particular processor. This pseudo-code (or *p-code*) is then interpreted by the run-time machine language program. This unusual approach to compiler design allows IC to offer the following design tradeoffs:

- Interpreted execution that allows run-time error checking and prevents crashing. For example, IC does array bounds checking at run-time to protect against programming errors.

- Ease of design. Writing a compiler for a stack machine is significantly easier than writing one for a typical processor. Since IC's p-code is machine-independent, porting IC to another processor entails rewriting the p-code interpreter, rather than changing the compiler.

- Small object code. Stack machine code tends to be smaller than a native code representation.

- Multi-tasking. Because the pseudo-code is fully stack-based, a process's state is defined solely by its stack and its program counter. It is thus easy to task-switch simply by loading a new stack pointer and program counter. This task-switching is handled by the run-time module, not by the compiler. Since IC's

ultimate performance is limited by the fact that its output p-code is interpreted, these advantages are taken at the expense of raw execution speed.

3.9. Motor Interface

Using a separate motor power source The Handy Board's internal battery provides a 9.6V power supply for operating DC motors. This is generally adequate for powering motors rated between six and 12 volts. Six volts motors will run somewhat "hot" with the internal battery supply. A separate motor battery can provide any voltage from about six to 36 V.

The procedure requires cutting the thick motor power trace and then plugging in the new motor battery into the motor power header. Figure 3.19 shows where to cut the motor power trace. The shown input part in Figure 3.19 has four pins. These inputs are "++--" from left to right. This order should not be reversed, else the board will burn up within a couple of seconds.

This strategy was used to supply power to the motors, because the battery of the controller was not sufficient to support the motors long enough to cover all area of a house. The NiCd batteries as mentioned previously were added two pins on each output and fed into the circuitry.

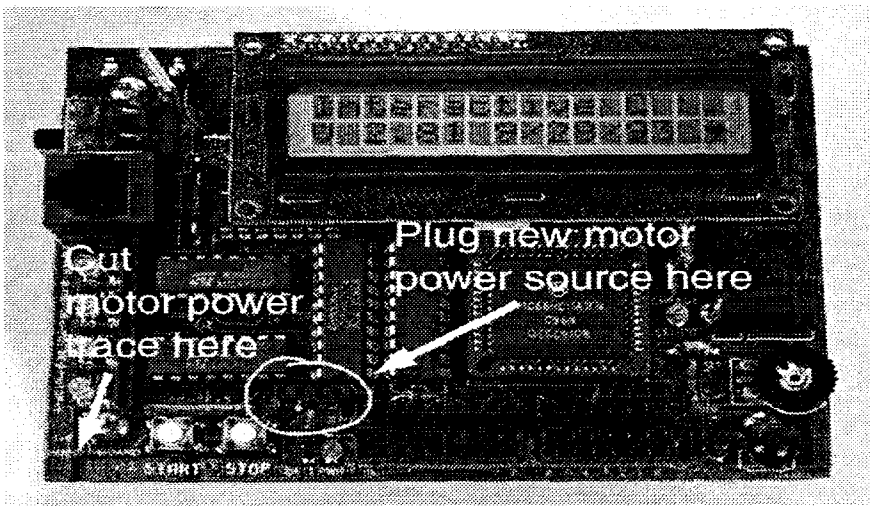


FIGURE 3.19. Handy Board motor interface.

3.10. Glue Selection

As throughout the steps of the design, a lot of things would change and during these changes a lot of mistakes would be done. Thus using silicon as glue, that is strong enough to attach equipment to the platform especially bumpers and motors, but weak enough to break apart by hand and that would not be effected by collisions, would be an appropriate choice. Using a heater pistol, some silicon was melted and was used to attach parts. Since it cools immediately, it does not harm the electronic parts.

3.11. The Vacuum Cleaner's Program

In the control program many behaviours, all running in parallel on a small microprocessor that is inherently a sequential machine is implemented by multitasking. Multitasking is to run a loop that, when repeated, gives a small amount of time to each behaviour. In this way, we simulated the effect of all behaviours running simultaneously. This is because of "start_process" command of IC. Our approach is somewhat a combination of the traditional and the subsumption approaches, which uses implicit invocation. Our architecture is based on hierarchies of tasks, the task trees. Figure 3.20 shows a sample task tree.

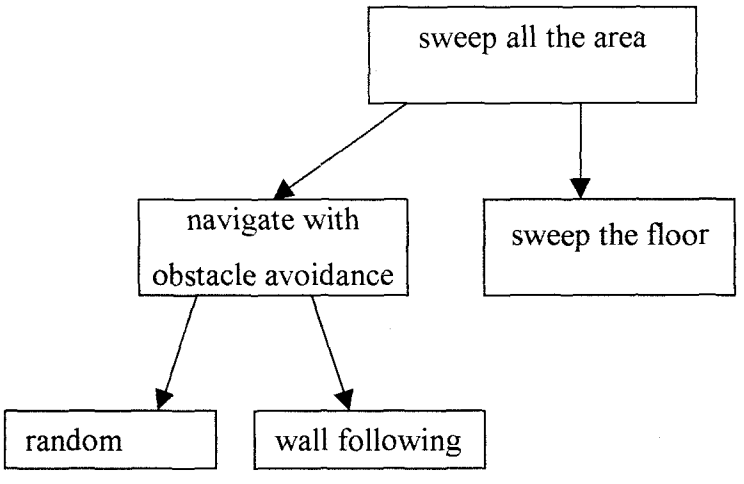


FIGURE 3.20. A sample task tree.

Parent tasks initiate child tasks. We, as designer of the software, defined temporal dependencies between pairs of tasks. An example temporal constraint is: "A must complete

before B starts." (e.g. the wall alignment must complete before wall following starts.) These features permit the specification of selective concurrency.

In our approach rather than differentiating each behaviour separately, we preferred to group some. We preferred to put obstacle avoidance into one, wall following into one, beacon detection mode into one and wall aligning into one. In the obstacle avoidance process, we implicitly used the decision mechanism using else-if sequences, which directs the messages to tasks. This approach contains the following features:

- Exceptions: Certain conditions cause the execution of an associated exception handler. Exceptions override the currently executing task that causes the exception. They quickly change the processing mode of the robot and are thus better suited for managing spontaneous events (such as a floor absence, which is a dangerous change in terrain.)

- Superimposition: Tasks can be intercepted by routines superimposed on an existing architecture, i.e., task tree. For instance, a safety check procedure can use this feature to validate all outgoing motion commands (e.g. the collision detection is made on every move toward a goal like aligning to a wall.)

- Monitors: Monitors read information and execute some action if the data fulfill a certain criterion. This feature offers a convenient way of dealing with a lot of routines by setting aside agents to supervise the system.

This approach has the following advantages:

1. Combinational behaviours could be implemented.
2. Since the approach is not a plan once and run forever one, we could easily change the design.
3. It is as fast as the subsumption architecture and needs the same resources.
4. It permits a clear-cut separation of action and reaction.
5. It incorporates concurrent agents in its model. It is evident that multiple actions can proceed at the same time, more or less independently.

6. It makes incremental development and replacement of components straightforward. It is often sufficient to register new handlers or exceptions, no existing components feel the impact.

The first advantage is a very crucial one. When the robot approaches a wall it may also want to check whether its left or right is empty or not or better whether it is in a claustrophobic environment.

Even more interesting than the speed and space advantages of this architecture are the possibilities of what this paradigm might hint to us about models of intelligence. Because seemingly complex behaviours can be seen to emerge from what we know are very simple reflexive behaviours, perhaps so complex that we hypothesis exist in what we acknowledge as intelligence might actually just be combinations of much simpler mechanisms. Let's imagine that vacuum cleaner moving in a room. We, as humans, see walls, chairs and table lamps. These images bring to mind the association that people sit in chairs and that table lamps are useful for reading. Vacuum cleaner sees none of these things yet can operate effectively in this environment.

In summary, this approach offers a comprehensive set of features for coordinating the tasks of a robot while respecting the quality and ease of development requirements. The richness of the scheme makes it most appropriate for our robot and more complex robot projects.

3.11.1. Autonomous Vacuum Cleaning Agent

The assumed goal is to develop an autonomous mobile agent for vacuuming enclosed areas with obstacles. The next few paragraph details further assumptions and approach taken in this work toward the realisation of the stated goal. Detailed algorithms will be presented in the subsequent sections of the thesis.

3.11.2. Assumptions about the Environment

We assumed that the environment is closed-off, the stairs appears like floor absence that the robot should not surpass and the furniture in the room appears like obstacles to the proximity sensors. A vacuum cleaning robot should, of course, clean all horizontal areas in the room not occupied by furniture. The model does not include the furniture moving policy and so those areas will not be touched. The robot should not damage furniture while performing its action and should optimise the parameters of time, energy consumption and capital investment.

3.11.3. Primitive Behaviours

Machine behaviours constitute the central focus of this research. The following behaviours were taken as primitives.

3.11.3.1. Obstacle Avoidance. For collision avoidance IR sensor and tactiles were employed. Obstacle detection derives from the sensor-based conditional: "if infrared sensors sense an object in their territories or any of the bumpers sense a collision then the obstacle flag is set to true."

3.11.3.2. Floor Detection. Floor detection was implemented for stairs, holes and for such cases that the floor does not exists. During the tests this capability helped the robot to sweep in presence of wheel chairs, hills and small obstacles without falling off. Such an instance is shown in Figure 3.21. In this figure the robot is on the top of a hill, it will not go any further but turn back and cruise in another direction.

3.11.3.3. Random Sweep. Random Sweep constitutes the base area coverage algorithm. Essentially the robot moves forward until an obstacle is detected, at which time it turns at a randomly chosen time between zero and five seconds and randomly chosen side, left or right. Figure 3.22 presents the pseudocode of the algorithm. This algorithm will be a measure for any sophisticated algorithm to be implemented on the robot.

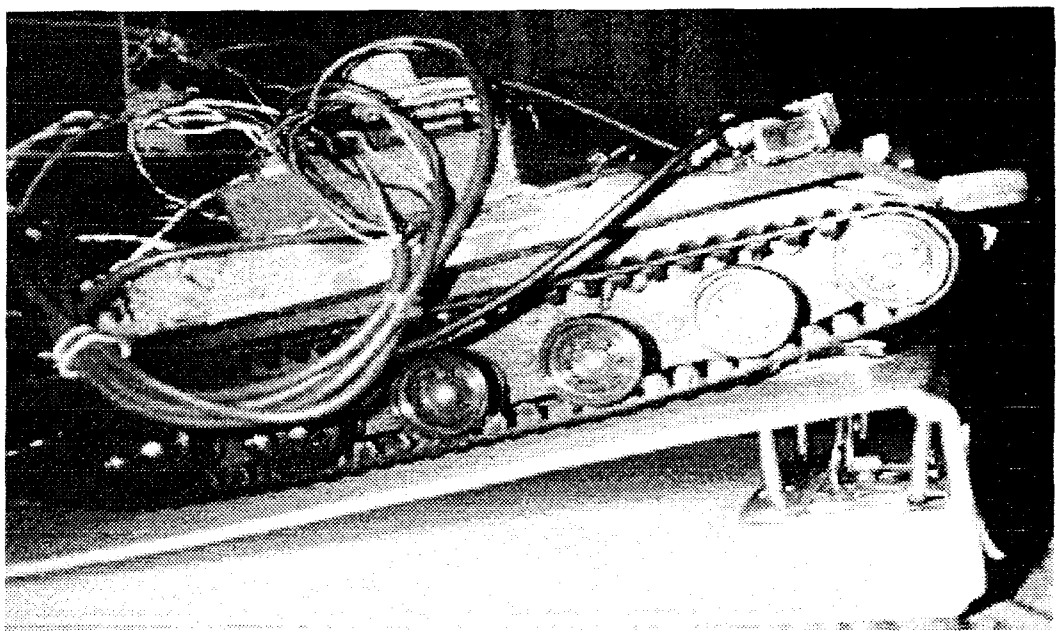


FIGURE 3.21. Floor detection helps the robot to sweep hills.

3.11.3.4. Wall Following. Side viewing IR proximity detectors allow the robot to follow walls. The robot attempts to stay at a fixed distance from the wall for a random period of time. Wall following permits the robot to circle furniture and follow along the room walls. The walls the robot follows can be on either on the right or left side of the robot.

```
/* Random Sweep*/
do{
   if obstacle = false then go forward
  else {
        choose Random(Left,Right)
          if chosen = Left then
               turn_to_left Random_Uniform(0,5)
          else
               turn_to_right Random_Uniform(0,5)
        }
}
```

FIGURE 3.22. Random sweep algorithm.

3.11.3.5. Docking. After a period of cleaning the robot begins to search for its docking area. It stays here for a period sufficient to recharge its batteries, then continues its work.

3.11.3.6. Claustrophobia. This behaviour enables the robot to break out from confined areas. The objective is to keep the robot from spinning around in place in confined areas. If the robot detects objects at its sides and front, it switches to claustrophobic mode and it moves backward until it breaks out. The claustrophobic state is set whenever the front, left and right sensors sense an object in their sense territories.

3.11.3.7. Combined Behaviours. The primitive behaviours alone do not adequately provide area coverage, although random sweep alone provides impressive coverage. Here, the definition of combined behaviours, which are tested, is given.

3.11.4. Control Software Module Definitions

To form a trivial basis, we have implemented a random sweep algorithm and a wall following routine. For any algorithm to be implemented further, one will compare the computational complexity and performance with the random sweep's computational complexity and performance.

In order to have the robot navigate in an unknown environment the robot at least must be able to perform obstacle avoidance and floor detection. Both of the implemented algorithms perform these behaviours. The wall following behaviour is added in order to help an exploration function to be added in the future and to sweep around objects close to walls. The algorithm's modules are as follows:

3.11.4.1. Random Number Generator. This routine returns a random number with a given seed as input.

3.11.4.2. Sense. This is actually an infinite loop process. It checks every sensor every 0.1 seconds and sets appropriate flags. This process helps the programmer to be able to change the sensor and motor ports on the board easily without changing the rest of the software.

3.11.4.3. Docking Area. This is also implemented as a process. It helps the robot to park itself in front of an active beacon mounted on a wall. The robot has a photocell to detect

the beacon in the front. Therefore, it will try to park itself from the front. The pseudocode is as in Figure 3.23.

```
if robot is in docking area search mode then
    kill all active processes
    repeat
        go forth turn left until beacon is seen
        go toward beacon for some time
  until an obstacle in the front is sensed
stop all the motors including the fan
```

FIGURE 3.23. Docking area search algorithm.

3.11.4.4. Turn Random Uniform. This routine turns the robot to the left or to the right as indicated by its only argument. The robot turns to the specified direction for an interval that is assigned randomly between two seconds to seven seconds. The random selections constitute a uniform distribution between two and seven. The turn operation is a function of time since representing it as a function of degrees is not accurate in most cases in a real environment because of slippage, coefficient of friction of the floor especially heterogeneous driving when one of the palettes are on a carpet and the other is not.

3.11.4.5. Claustrophobia. This process helps the robot to break out of confined areas. Confined areas are detected only when the front and the side infrared sensors detect an obstacle at the same time. The robot moves backward until its either side is empty. Left side is preferred if both sides are empty. This process runs in parallel to the obstacle avoidance process. The pseudocode is as in Figure 3.24.

```
while both sides are full
    go back
if left is full then
    turn right
else
    turn left
```

FIGURE 3.24. Claustrophobia algorithm.

3.11.4.6. Check Bumpers. This routine handles collision detection in the wall following mode. The direction of the wall following is passed as an argument to the routine. By the direction, the side where the wall is to be found is implied. The pseudocode is as in Figure 3.25.

```
if  robot bumped its Front Right then
        go back
        turn opposite to the wall follow direction
if robot bumped its Front Left then
        go back
        turn along the wall follow direction
if robot bumped its Rear Left then
        go forth
        turn opposite to the wall follow direction
if robot bumped its Rear Right then
        go forth
        turn along the wall follow direction
```

FIGURE 3.25. Check bumpers algorithm.

3.11.4.7. Align Wall. This process continues to turn to the wall follow direction until it finds a wall. This process runs in parallel to other algorithms, thus obstacle avoidance is preserved. The pseudocode is as in Figure 3.26.

```
while in wall follow mode
        if  there is an obstacle in the front and wall follow direction is full then
                corner is detected so turn in the opposite direction of wall follow
                direction until there is no obstacle in the front
                continue turning in the same direction until wall is seen again
        if no wall in the wall follow direction
                        turn to wall follow direction
```

FIGURE 3.26. Align wall algorithm.

3.11.4.8. Wall Follow. This function determines the duration of the wall follow mode and the wall follow direction. The pseudocode is as in Figure 3.27.

```
if  one of the sides is full then
        choose the empty side
else
        make a random decision with probability p (=50 per cent)
turn in the opposite direction of the wall follow mode till a wall is seen.
start align wall process
wait sometime and exit wall follow mode
```

FIGURE 3.27. Wall follow algorithm.

3.11.4.9. Avoid Obstacles. This process consists of the main functionality of the robot. It acts as a controller and coordinator telling the robot when to follow walls or what to do in case of a collision, etc. The inputs from sensors are put into a hierarchy and then a decision about the movement is made. The highest priority in the hierarchical tree is given to bumpers, next there is the floor sensor and finally the front sensor. The pseudocode is as stated in Figure 3.28.

```
if a collision occurs then
        turn to the opposite direction of collision
else  if no floor in the front then
    if both sides are full then
            start claustrophobia process
    else
            go back and turn to a random side uniformly
else if there is an obstacle in the front then
        start wall following process with 50 per cent probability or turn
        left randomly
```

FIGURE 3.28. Avoid obstacles algorithm.

4. EVALUATION OF PERFORMANCE

The performance tests of the robot were done in two ways. First one was to write a simulator and test the results. Second one was to observe the real behaviours of the robot. The second approach is time consuming and also it is not an easy task to setup various environments for the robot. The first approach saves time and helps to design different environment conditions, but it does not allow fully implementing the actual behaviours in some circumstances. Using the first approach, area coverage and efficiency metrics, which are explained in the subsequent sections, were measured. Using the second approach robot's behaviours in some typical circumstances were observed.

This part will briefly mention:

1. RoboSIM, a robot simulator software.
2. The test results obtained by the simulator and real experiments.

4.1. Metrics

For a vacuum cleaner robot, one has to think of two basic metrics in terms of its software. Then, the performance of the cleaner can be measured by some function of these two metrics. The metrics are:

- Area coverage: This metric defines how much of the space available for vacuum cleaning is swept. The robot should be able to sweep at least 95 per cent of the total area to be vacuumed.

- Efficiency: This is actually the performance of the area coverage algorithm. This metric defines how many times the robot passes through a point. 50 per cent efficiency implies that the robot covers every point twice on the average. Positively one may consider sweeping a point twice as a normal sweeping behaviour, so the efficiency would be considered as 100 per cent.

4.2. RoboSIM - A Robot Simulator Software

4.2.1. The Model

The robot has four bumpers and three infrared sensors. The robot is represented as an ellipse and each quadrant forms up a bump sense territory. This does not contradictory to the original implementation of the robot, since the bumpers' design allow such an approximation (The bumpers do not imply a tiny touch point.) Whenever one of the sensors senses an object, the corresponding sensor is lighted and a good visual effect is obtained. Moreover, this way a better debugging of the robot's movement is obtained.

4.2.2. Functionality

The simulator has only one central screen where all the simulation functions can be carried out and co-operative screens where parameters to the main screen are supplied. Currently the program deals with the random sweep and random sweep with wall following algorithms. The probability of following a wall is also parametric, thus more predictive results may be obtained.

The simulator is used to the area coverage and efficiency metrics. To accomplish this, at each time step the robot leaves a footprint on the screen during the simulation. In the end, the white spaces constitute the uncovered areas while black areas constitute the objects and the walls. Since the step count is known these metrics can be calculated easily.

$$\begin{aligned} Eff &= A(S_c - 1) + (A_r / A_t) * 100 \\ C &= (A_t - A_o - A_w)/A_t * 100 \end{aligned} \qquad (4.1)$$

where Eff is the efficiency, A is the area covered in one move, S_c is the step count, A_r is the robot's area, A_t is the total area, C is the coverage, A_o is the object's area and A_w is the total white area.

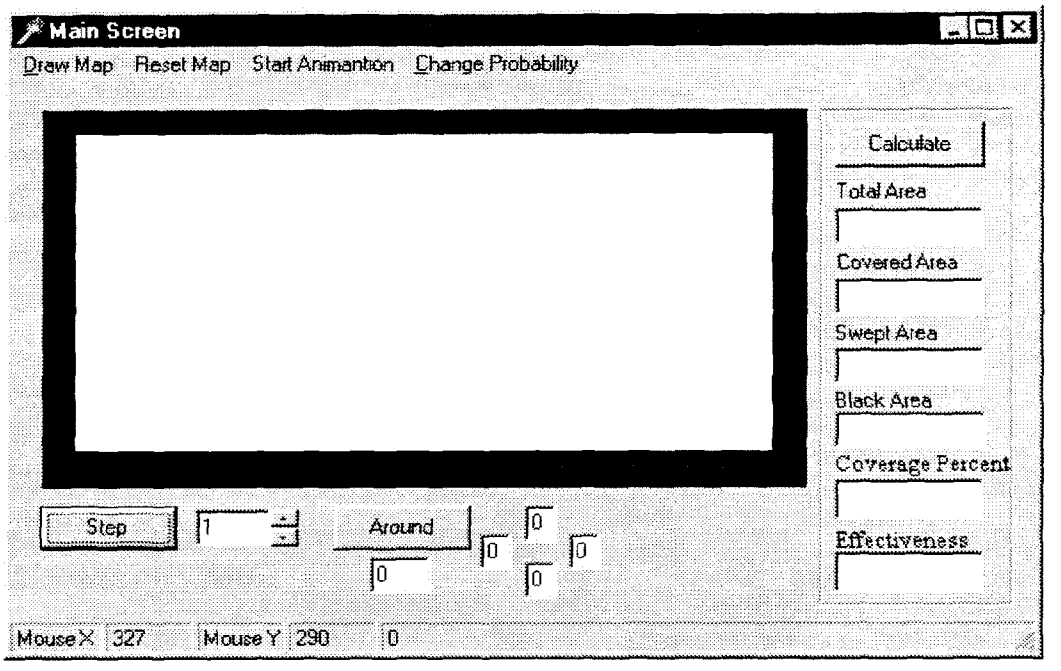


FIGURE 4.1. Initial screen.

The screen is shown in Figure 4.1. In the middle is an empty room, shown in white, where the robot sweeps.

The menu consists of the following items:

- Draw Map: The obstacles are added using another screen. By this approach, the modularity of the screen enables one to test the environment on different room setups.

- Reset Map: Clears the map and the obstacles.

- Start Animation: Starts the animation of the robot, with a given position, direction and the unit time the animation will be performed. To conduct series of tests an array of 20 initial data is also suggested in the subsequent screen.

- Change Probability: The wall following mode's probability is given. 0 per cent implies no wall following but pure random sweep.

When the "Draw Map" selection is selected, a screen appears as in Figure 4.2 where the position and the size of the obstacle are asked. The sample input screen is shown in Figure 4.2.

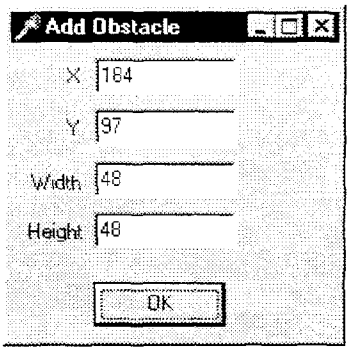


FIGURE 4.2. Sample obstacle adding screen.

The obstacles are considered to be rectangles. The maximum number of obstacles is 100 for efficiency but this may be increased if necessary. As a result of such an obstacle adding, screen will look like as in Figure 4.2.

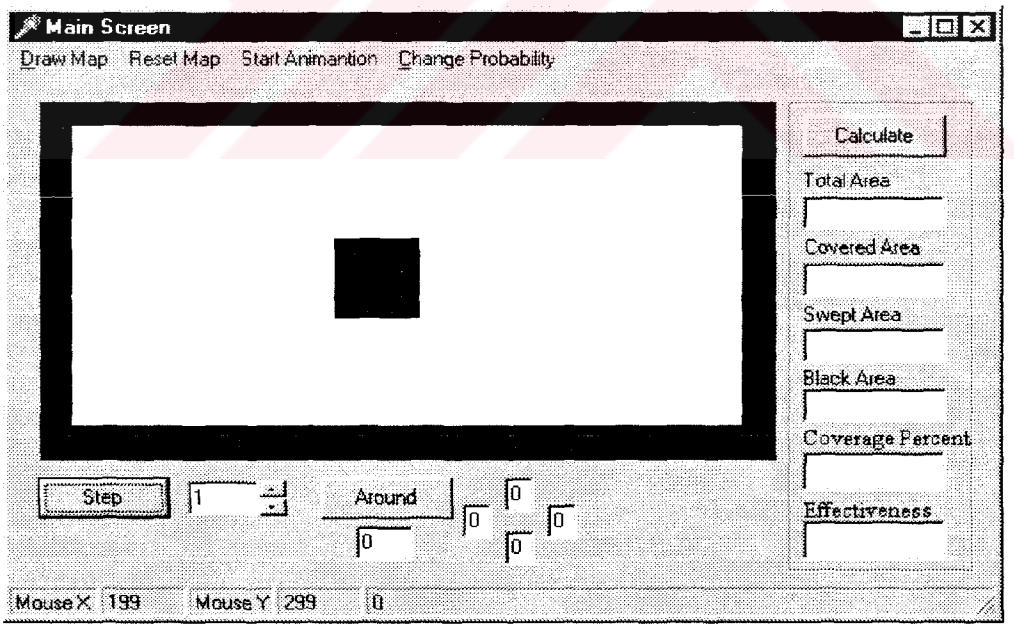


FIGURE 4.3. Screen after an addition of an obstacle.

When the "Start Animation" item is selected, a screen appears asking the initial robot position, direction and the number of simulation steps where the robot moves one forth of its size in each step. The screen that asks for these is given in Figure 4.4.

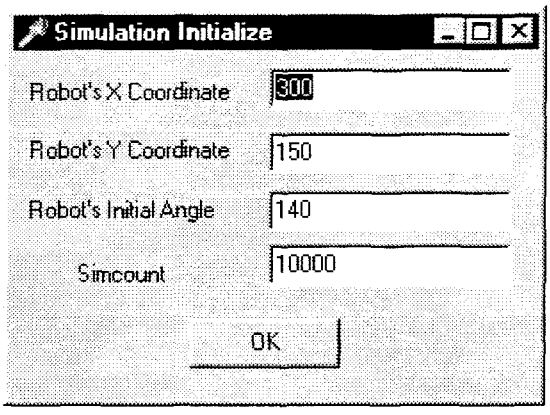


FIGURE 4.4. Simulation starting screen.

A sample ten step simulation screen is given in Figure 4.5.

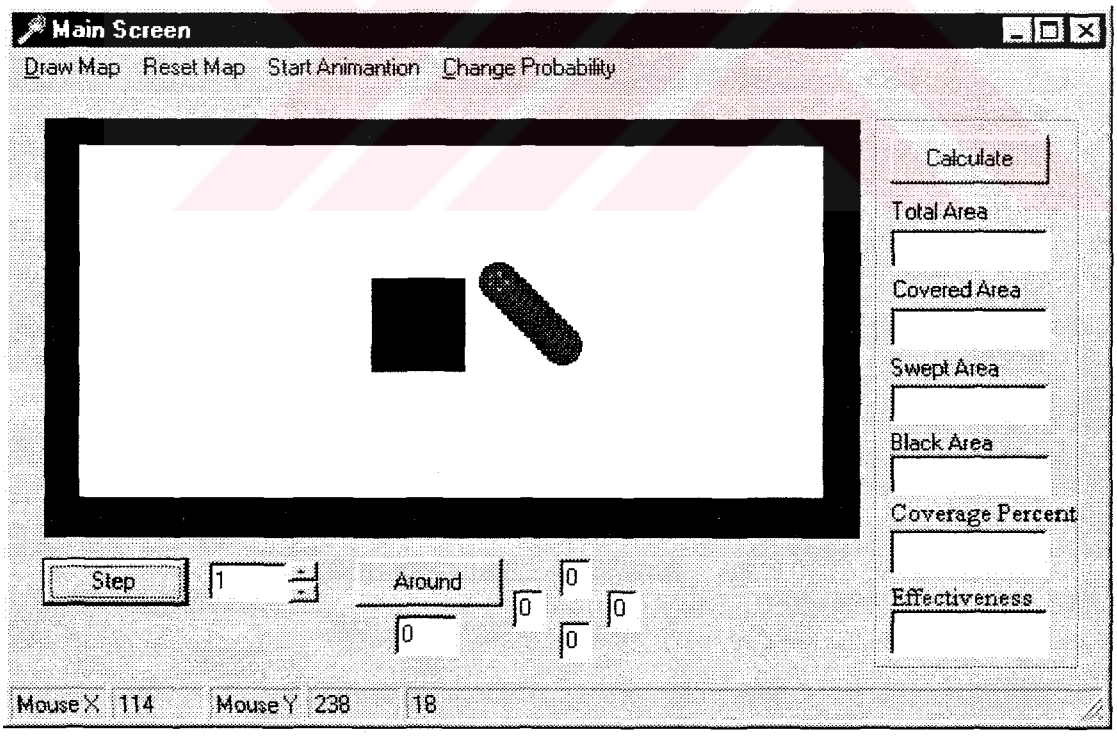


FIGURE 4.5. Sample 10 step simulation.

The "Step" button on the main screen facilities the ease of performing the simulation step by step. The spinner next to it defines how much simulation steps the robot

will take. The "Around" button helps one to get territory information of the robot. When it is clicked the four boxes on the right defines if the sensor looks north, south, east or west, would it see the environment. This is just to test the infrared module's working properly. This module can not be implemented on the robot. Moreover, the box below the button will give the direction of the robot in degrees.

Such an infrared sensing step is shown in Figure 4.6. Note that the Around button yields the correct result so the infrared is functioning normally. Also note that the infrared in the front is lighted.

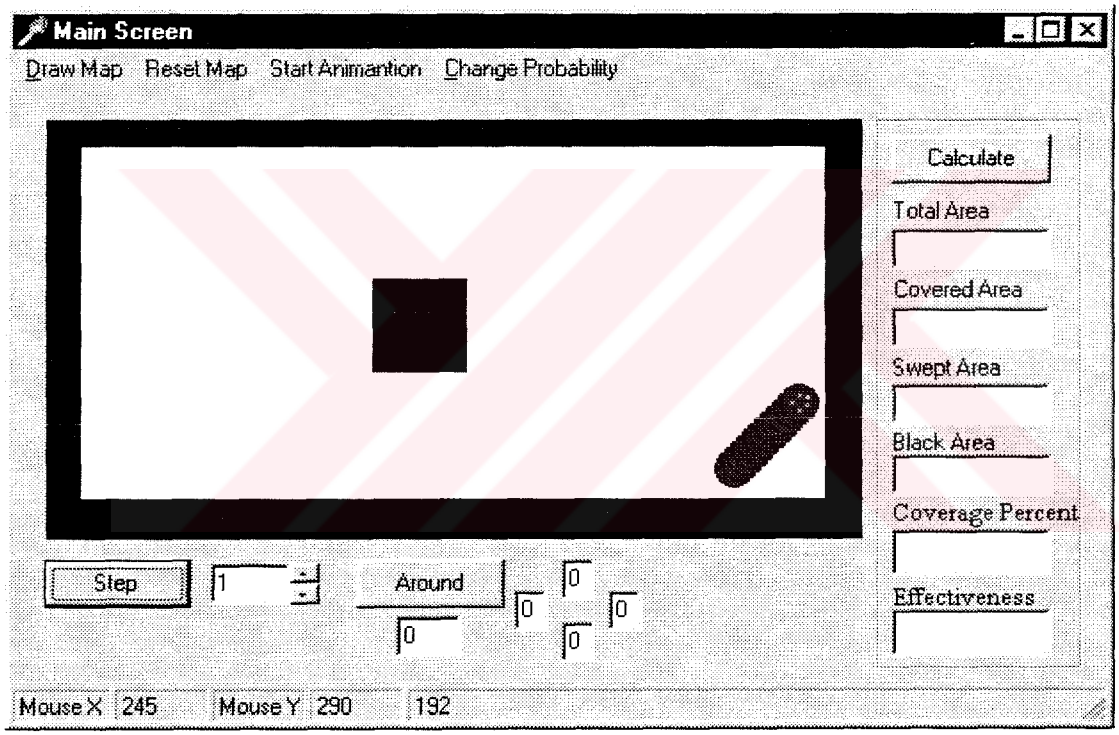


FIGURE 4.6. Infrared in the front detects an obstacle.

Likewise whenever the bumpers hit, they are lighted as well.

"Change Probability" menu item opens a dialog screen where the wall following probability is given. The appearance of the screen is as in Figure 4.7.

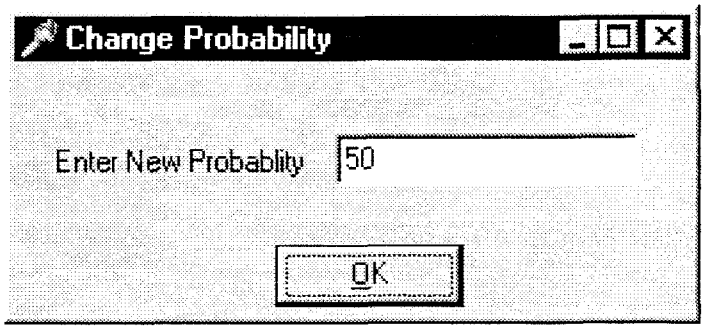


FIGURE 4.7. Wall following probability is entered here.

Zero probability denotes no wall following but pure random sweep. A wall following behaviour is exampled in Figure 4.8.

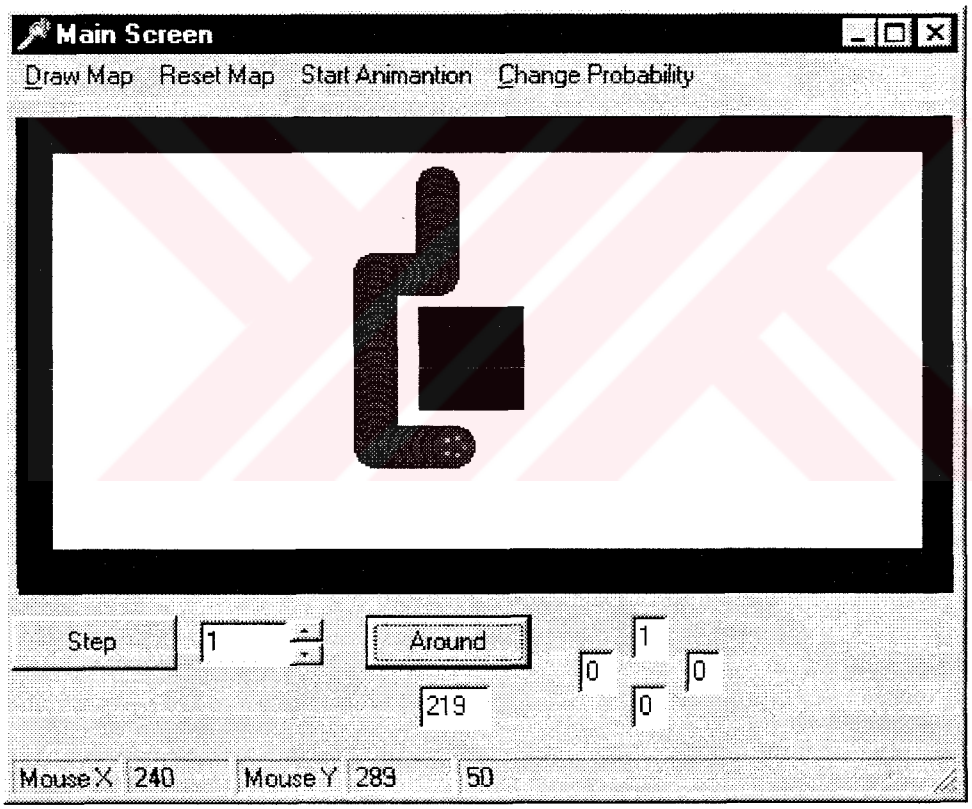


FIGURE 4.8. Wall following behaviour.

4.3. Test Results

The two algorithms, random sweep and random sweep with wall following, were tested.

4.3.1. RoboSIM results

In the beginning, a test set of twenty initial positions representing typical situations were created. An initial point consisted of robot's x-coordinate, robot's y-coordinate relative to screen's top left corner and robot's angle calculated as counter-clockwise and relative to east. Secondly, a test set of scattered obstacles was created. An obstacle consisted of its top left corner's x and y coordinates relative to screen's top corner and its height and width. For each of the object arrangements, twenty initial positions are tested, the average and standard deviation of these twenty results are noted.

The resulting data are given in TABLE 4.1.

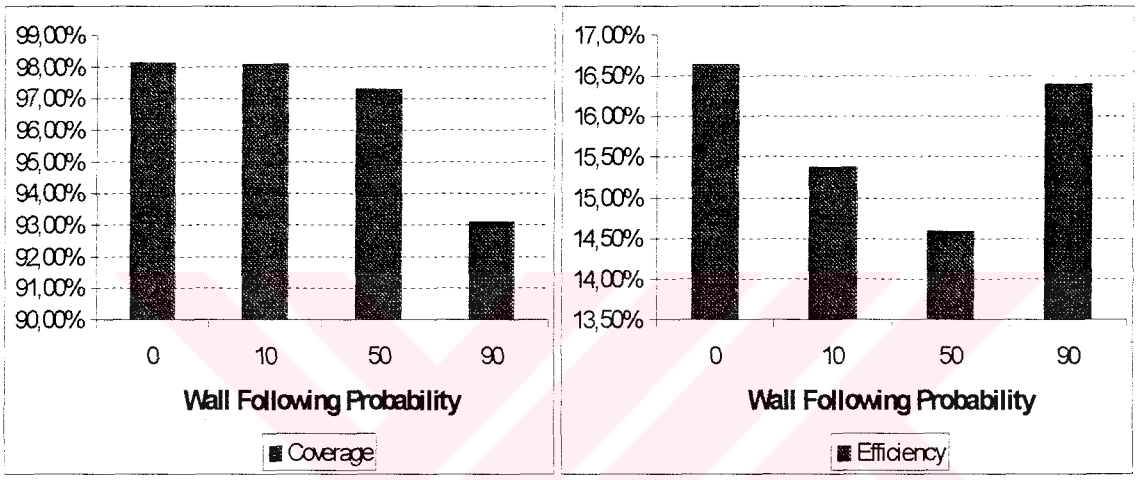
TABLE 4.1. Average of every test set in RoboSIM.

Probability	Obstacle	Empty	Swept	Covered	Coverage	Efficiency
0	21183	67841	66579	400236	98.14%	16.63%
10	21183	67841	66547	432676	98.09%	15.38%
50	21183	67841	66014	452324	97.31%	14.59%
90	21183	67841	63171	384948	93.12%	16.41%
0	23487	65537	63818	400236	97.38%	15.95%
10	23487	65537	63570	432676	97.00%	14.69%
50	23487	65537	62330	452324	95.11%	13.78%
90	23487	65537	60791	384948	92.76%	15.79%
0	29327	59697	56762	400236	95.08%	14.18%
10	29327	59697	56883	400604	95.29%	14.20%
50	29327	59697	56452	402068	94.56%	14.04%
90	29327	59697	55821	400972	93.51%	13.92%

The probability column defines the wall following probability. The obstacle and empty columns give space covered by obstacles plus walls and the area covered by empty respectively. The swept column gives how much of the empty space is covered by the robot at the end of 5000 simulation steps. The covered column gives the total area the robot swept at the end of 5000 simulation steps. These columns values are derived from the

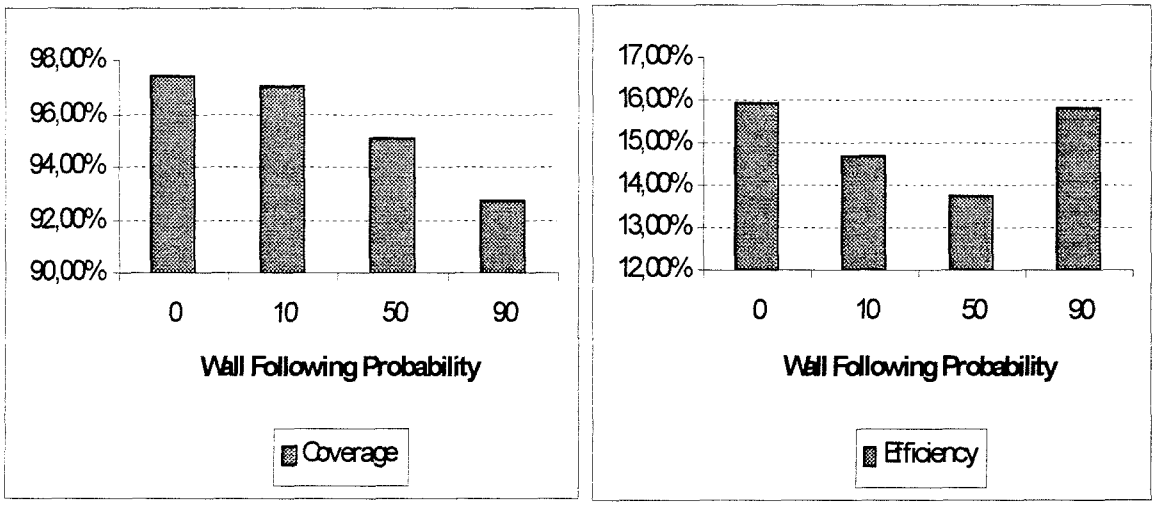
result sets obtained by applying the twenty initial points' set. The last two columns are the values of the calculated metrics.

The charts obtained by these results, grouped according to area of obstacles are given in Figure 4.9 , 4.10 and 4.11. As seen the probability of wall following does not effect the performance very much. On the other hand the trade off between coverage and efficiency is seen from these figures.



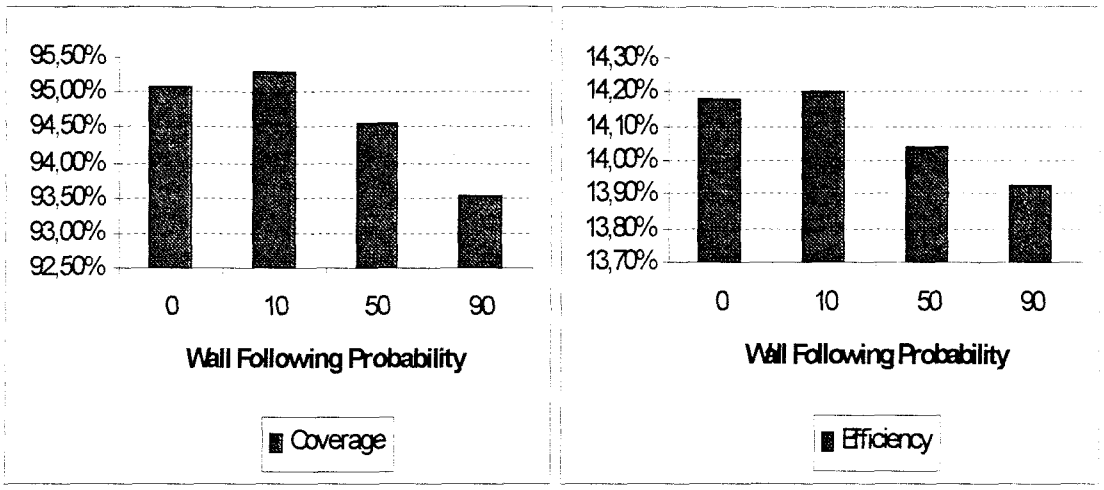
(a) (b)

FIGURE 4.9. No obstacle. (a) Coverage statistics. (b) Efficiency statistics.



(a) (b)

FIGURE 4.10. One obstacle. (a) Coverage statistics. (b) Efficiency statistics.



(a) (b)

FIGURE 4.11. Three obstacles. (a) Coverage statistics. (b) Efficiency statistics.

So, a wall following routine can be used for navigation or exploration purposes without effecting the overall performance very much. One should also note that as there are more obstacles scattered around, wall following with a low probability, increases the performance. Wall following behaviour helps the robot to explore the areas of the room, which is nearly impossible with pure random sweep. Although a very low efficiency level is achieved with these algorithms, one may consider that covering each point twice, a good sweeping factor which is analogous to human vacuum cleaning habit. In this case, the calculated efficiency should be multiplied by two.

Noting that these results are calculated in an environment where there is no slippage, no small obstacles with one or two centimeters height, no legged chairs, no level change in the surface and all obstacles are identical, the results reflect a rough estimation. As will be mentioned in the subsequent part, there are many exceptions to the simulator's results because of the complexity of a real indoor environment. One example of such an exception is that the infrareds' sense territories are not constant. They are highly dependent on the lighting of the room, which is not again homogeneous, the colour of the walls, the floor and the obstacles, which may even constitute black and white patterns.

4.3.2. Real Experiments

Since the simulator was inadequate to know that the robot could be used indoors, real experiments were made. As the robot was tested individually the following behaviours were observed:

- The robot could easily break out of confined areas. For sufficiently wide areas the robot could follow the confined walls else it performed claustrophobic behaviour. By sufficient, minimum eleven tenth of robot's size is implied. Figure 4.12 shows the boundary of the robot in such a circumstance.

- The robot was able to follow walls. Figure 4.12 examples such a behaviour.

- The robot did not fall from stairs, tables and such high places using only one IR sensor.

- The robot could climb hills up to 40 degrees.

- The robot could not go through messy places such as vast amount of wires, which are common in every house.

- The robot could not handle furniture that have empty spaces underneath where the empty spaces' height is bigger then IR's distances from floor but smaller than robot's height.

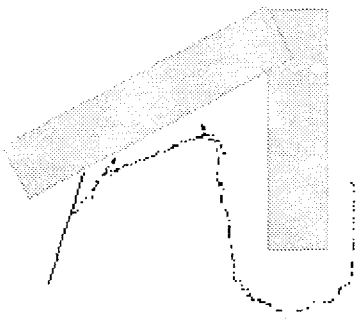


FIGURE 4.12. Boundary of the robot's path in a confined area.

- Poor reflection of light from dark surfaces often caused the robot to feel "empty floor" and rarely caused the robot to bump into walls.

- The robot moved onto obstacles with a very small height and returned backwards and turned in some direction. Therefore the robot could easily handle wheel chairs, some tables, etc.

- As curtains that touches floor appeared as obstacle, they did not constitute problems.

5. CONCLUSION

We approached the problem of autonomous vacuuming from a low-level standpoint, using a robot equipped with simple, low-cost sensors and an eight bit microcontroller with 32K of RAM for software. We used the robot simulator to measure the vacuum cleaner's efficiency. Each test recorded the movement of the robot within a specific count of robot's move.

According to our simulator, a purely random sweep of movement covers the room with a better efficiency than random sweep with wall following in environments, which has small number of obstacles. Unfortunately, most domestic apartments have lots of furniture. The addition of a wall following behaviour, triggered with probability 0.10 at each obstacle detection, seems to increase performance about efficiency in environments, which have a lot of furniture. Increasing the sensitivity of the object detectors traps the robot in wide open spaces. Decreasing the sensitivity of the object detectors allows the robot to wander through narrow corridors into new open spaces. The wall following behaviour also seems to "drag" the robot to distant parts of the room. In an attempt to mix thorough open space coverage with full-room coverage, we combined these behaviours. This strategy provided coverage that was marginally better than the random walk behaviour. We implemented a claustrophobia indicator that successfully freed the robot from narrow corridors and corners.

The autonomous vacuum cleaner is not ready for commercialisation. Nevertheless, many of the achieved results are very promising. The combination of the robot's shape, its sensor system and its algorithm play well together and make the task of cleaning an unknown and unstructured environment feasible.

The developed robot is capable of dealing with a real environment in real-time. Even in complex environments, this robot could fulfil its task because of its powerful algorithm.

To reduce the difference between the efficiency of area coverage with the techniques developed here and the better efficiency of a human with a vacuum cleaner,

will probably require more sophisticated sensors, more resources, short-term memory based strategies and sensors providing feedback information about how successfully the task is being performed.

APPENDIX A. RoboSIM SOFTWARE

The modules used in the simulator are shortly described in this section. These module definitions may give a clue on how the simulation is performed theoretically and how successful such a simulator can implement the actual behaviours of the vacuum cleaning robot.

Procedure drawSensor (x, y, ssize :integer; scolor :TColor): x and y constitute the sensor's centre point. ssize is the sensor's radius. Scolor is the colour of the sensor. Using these arguments a circle representing a sensor is drawn. Whenever a sensor is inactive it is drawn with a matte colour while is active it is drawn with an active colour.

Procedure DrawRobot (bumpTL, bumpTR, bumpBL, bumpBR, IRF :boolean): Robot, bumpers and the IR sensors are drawn. Each flag that are arguments to the procedure states whether the sensor is active or not.

Function clipobstacle (x1, y1, x2, y2 :integer; obstacle:TShape) :boolean: A line segment with end points (x1,y1) and (x2,y2) is clipped with an obstacle. If they correspond then function returns true. This procedure is in sensor activation. The infrared sense territory, which is a global to the simulator, is assumed to form a line segment, in the robot's direction of movement.

Function between (test, first, second :integer) :boolean: This function returns if an integer, test, is between the integers first & second. This procedure is highly used in clipping algorithms.

Function dist_bet_points (x1, y1, x2, y2 :integer) :real: This function returns the distance between the two points defined by (x1,y1) and (x2,y2).

Function bump (obst :TShape) :string: This function returns true if the robot has bumped into the obstacle obst.

Function IRFront :boolean: This function returns if IR sensor in the front senses an object. The distance between the robot's centre and the obstacle's each edge is measured and compared with the robot's radius.

Function check(dir:integer):boolean: This function is used for debug use only. At any time if the user clicks on the around button, this function called and checked if there is an obstacle with a distance of IRSenseTerritory in the direction "dir."

Procedure go(steps:integer): In real robot algorithm, the vacuum cleaner robot show characteristic behaviours (wall following, claustrophobia...) for some steps which can be random or until some conditions are satisfied. This procedure is the simulator implementation of these moves in virtual room.

Procedure wall_follow: The robot tries to follow wall. It continuously checks whether its follow mode is satisfied. Many of decisions necessary to follow wall are taken. The robot should understand when it comes to a corner if it's convex or concave and in every move it should continuously check claustrophobia mode to break out the confined areas.

Function control_bumpers:string: Tactile sensors' data are retrieved. This function returns the name of the bumper by which the collision is detected. With these features, it is very helpful to take a decision about the next move.

Procedure nextmove: Main decision algorithm. This function is responsible for evaluating sensor inputs and with the aid of the functions mentioned above takes the decision about the mode of the next move (wall follow, random sweep ...). The procedure may also handle combined behaviours if necessary.

Procedure move: Simulation control algorithm. This function is responsible for drawing robot and simulation steps. in every step nextmove is called to take a decision about the next move.

APPENDIX B. PRICES OF COMPONENTS

Controller circuitry	250$	(x 1)	=	250$
IR Sensors	75$	(x 4)	=	300$
Photocell	75$	(x 1)	=	75$
Tactiles	0.5$	(x 4)	=	2$
Driving Motors	20$	(x 2)	=	40$
Fan Motor	4$	(x 1)	=	4$
Base	30$	(x 1)	=	30$
NiCd Battery	30$	(x 1)	=	30$
			Total:	**731$**

REFERENCES

1. Brooks, R. A., "A robust layered control system for a mobile robot," *IEEE Robotics and Automation,* Vol. RA-2, pp. 14-23, 1993.

2. Nelimzov, U. and T. Smithors, "Using motor actions for location recognition," *Proceedings of the First Europian Conference on Artificial Life,* pp. 96-104, Paris: MIT Press, 1991.

3. Steels, L., Building agents out of autonomous behaviour systems, In *The Biology and Technology of Intelligent Autonomous Agents.* NATO Advanced Study Institute, Toronto. Lecture Notes, 1993.

4. Brooks, R.A., "Elephants don't play chess," *Robotics and Autonomous Systems*, Vol. 6, pp.3-15, 1990.

5. Borenstein, J., H.R. Everett, and L. Feng, *"Where am I? Sensors and Motors for Mobile Robot Positioning,"* Massachutes: A K Peters, 1996.

6. Miller, D.P., R.S. Desai, E. Gat, R. Ivlev, and J. Loch, "Reactive Navigation through Rough Terrain: Experimental Results," *Proceedings of the 1992 AAAI Conference*, San Jose CA., pp. 823-828, 1992.

7. Engelberger, J.F., *"Robotics In Service,"* Massachutes: The MIT Press, 1989.

8. Jones, J., *"Mobile Robots: Inspiration to Implementation,"* Massachutes:A K Peters, 1993.

9. Pellerin, C., "The Service Robot Market," *Service Robots International Journal,* Vol: 3, pp.17-20, 1995.

10. Schofield, M., "Cleaning Robots," *Service Robots International Journal,* Vol. 1, pp.11-16, January 1990.

11. Serway, R.A., *"Physics,"* New York: Sanders College Publishing, 1990.

12. Martin, F., *"The 6.270 Robot Builder's Guide,"* Cambridge: MIT Media Laboratory, 1992.