ENVIRONMENT SENSING AND MODELLING, PATH FINDING
AND POSITION LOCATION FOR A MOBILE ROBOT

A Thesis submitted by
Richard Charles SEALS, BSc (Hons)
in partial fulfilment of the requirement
for the degree of
Doctor of Philosophy
of the University of Surrey

November 1985
ENVIRONMENT SENSING AND MODELLING, PATH FINDING AND POSITION LOCATION FOR A MOBILE ROBOT

BY Richard Charles SEALS

ABSTRACT

The results of previous research into mobile robotics were studied and several fundamental properties deduced. These were then considered in detail and optimum techniques selected for implementation on a prototype mobile robot. Several constraints were placed on the prototype mobile robot, such as the potential for untethered use (i.e. no umbilical connections) and minimum restrictions on the environment.

The investigation was concentrated in four main areas: environment modelling, path selection and following, absolute position location and environment sensors. An initial simulation was implemented on a mainframe computer which used an X-Y grid square model of the environment with a simple scanning rangefinder, to investigate the usefulness of the Means-end path finding algorithm. Results were obtained for varying states of the environment (i.e. known, unknown and partially known) which indicated that the path finding algorithm was suitable for implementation on the prototype mobile robot.

An improved environment model which used a quasi-continuous X-Y cartesian coordinate system was then constructed. This was designed to enable environment data from the simulated or prototype scanning rangefinders to be used as input to the control processes, with movement of the simulated or prototype mobile robot as the output. In this way the mobile robot was able to find paths using the model of the environment and then attempt to follow them in the physical environment. Alternatively, the prototype mobile robot was able to find a path through an unknown environment using data from the rangefinder only.

In addition, a position location process was implemented which operated by identifying the position of known objects in the environment by matching the distribution of three of the detected objects against the distribution of all known sets of three objects. Once the objects had been identified the position of the prototype mobile robot was calculated. The preliminary results indicated that the position could be found using this technique but that more investigation was required to reduce the ambiguity of the results.
ACKNOWLEDGEMENTS

The financial support provided by the Science and Engineering Research Council in the form of a Research Student grant and Conference expenses is acknowledged with gratitude.

The supervision of C. L. F. Haynes (now with ICL) and Dr Roger Fairwood is gratefully acknowledged.

My thanks also go to the members of staff and students of Middlesex Polytechnic.

This work is dedicated to the Glory of God and His son Jesus Christ.
CONTENTS

ABSTRACT ........................................................................................................... (i)

ACKNOWLEDGEMENTS ...................................................................................... (ii)

Chapter 1 : AN INTRODUCTION TO MOBILE ROBOTS................................. 1

1.1) HISTORY ........................................................................................................ 1
  1.1.1) What is a Mobile Robot ................................................................. 1
  1.1.2) Previous Research into Mobile Robotics ........................................ 2
    1.1.2.1) The NASA Mars Rover ......................................................... 3
    1.1.2.2) Artificial Intelligence ............................................................. 3
    1.1.2.3) Automated Guided Vehicle Systems ...................................... 4
    1.1.2.4) Mobile robotics .................................................................... 4

1.2) AN OUTLINE OF BASIC THEORY AND PROCESSES ....................... 4
  1.2.1) Fundamental control concepts of a mobile robot .............................. 5
  1.2.2) A self contained chassis ................................................................. 5
  1.2.3) Real-time control when moving in the environment ...................... 6
  1.2.4) Environment sensors ..................................................................... 7
  1.2.5) Modelling the environment ........................................................... 8
  1.2.6) Simulating the mobile robot .......................................................... 10
  1.2.7) Determination of the position of the mobile robot ......................... 11
  1.2.8) Path finding and following ............................................................. 12

1.3) APPLICATION AREAS OF RESEARCH .................................................. 13
  1.3.1) Mobile robotics .............................................................................. 13
  1.3.2) Robotics in general ....................................................................... 14
  1.3.3) Transportation (air, land and sea) .................................................. 14
  1.3.4) Environment sensing .................................................................... 15

1.4) THE OBJECTIVES OF THE RESEARCH ............................................... 16
  1.4.1) The path finding and following techniques ................................... 16
  1.4.2) The construction of environment models from sensor data ............. 16
  1.4.3) The determination of the position of the mobile robot ................... 16

1.5) SUMMARY OF THE INTRODUCTION ..................................................... 17

CHAPTER 2 : AN INVESTIGATION OF MOBILE ROBOTICS ......................... 18

2.1) MODELLING THE ENVIRONMENT ......................................................... 18
  2.1.1) The grid environment .................................................................. 19
  2.1.2) Listing the objects and/or the empty spaces .................................. 21
    2.1.2.1) Representation of objects as polygons or circles ....................... 23
  2.1.3) The mobile robot as a charged particle ......................................... 26
  2.1.4) Conclusion ..................................................................................... 27
  2.1.5) Proposed technique of modelling the
environment ........................................... 27

2.2) PATH SELECTION AND FOLLOWING ......................... 27
  2.2.1) Planning the path using the simulated environment ........................................... 28
    2.2.1.1) Planning the path through passageway mazes ........................................... 28
    2.2.1.2) Finding paths when the environment was modelled by circles ....................... 29
    2.2.1.3) Finding paths when the environment was modelled by polygons ..................... 31
    2.2.1.4) Path finding in a polygon model of the environment using truncated cones .......... 31
    2.2.1.5) The Means-end algorithm for finding paths ............................................. 32
    2.2.1.6) Conclusion ............................................. 33
    2.2.1.7) The proposed path finding technique ......................................................... 34
  2.2.2) Path following ............................................. 35
    2.2.2.1) The deviation of the mobile robot from the planned path ....................................... 36
    2.2.2.2) The effect of the mobile robot's physical characteristics on the path following problem ............................................. 36
    2.2.2.3) Conclusion ............................................. 38

2.3) LOCATING THE POSITION OF THE MOBILE ROBOT .......... 39
  2.3.1) Recognition of features in the environment ......... 39
    2.3.1.1) Feature detection using rangefinders ......................................................... 40
    2.3.1.2) Symbolic feature detection .......................................................... 41
    2.3.1.3) Conclusions ............................................. 41
    2.3.1.4) The proposed method of detecting features ............................................. 42
  2.3.2) The proposed method of finding the position of known features in the environment .......... 42
  2.3.3) The proposed method of calculating the position of the mobile robot ................. 46
    2.3.3.1) Accuracy of the position calculation process ......................................................... 49
  2.3.4) The estimation of position change of the mobile robot from a known starting point .... 51
    2.3.4.1) The calculation of change in position using assumed movement parameters .......... 51
    2.3.4.2) The calculation of the position change using internally measured movement parameters ......................................................... 52
    2.3.4.3) The calculation of change of position by measuring an absolute external quantity .... 53
    2.3.4.4) Correction of calculated position using the relative trends of the environment .... 54
    2.3.4.5) Conclusion ............................................. 55
    2.3.4.6) The proposed "assumed movement parameter" calibration process ......................................................... 56

2.4) ENVIRONMENT SENSORS ........................................... 56
  2.4.1) Rangefinders ............................................. 57
    2.4.1.1) The laser rangefinder .......................................................... 58
    2.4.1.2) The ultrasonic rangefinder .......................................................... 59
    2.4.1.3) The electromagnetic rangefinder .......................................................... 59
    2.4.1.4) The scanning rangefinder .......................................................... 60
4.3) PATH FINDING AND FOLLOWING WITH THE IMPROVED SIMULATED MOBILE ROBOT .................................................. 131
  4.3.1) Path finding in the simulated and physical environments .................................................. 131
  4.3.2) Examples of paths found ................................................................. 137
  4.3.3) Summary .................................................................................. 143

4.4) FINDING THE POSITION OF THE MOBILE ROBOT .............. 145
  4.4.1) The implementation of the position finding process ............................................................................. 145
  4.4.2) Examples of finding the position of the mobile robot .............................................................................. 150
  4.4.3) Summary .................................................................................. 160

CHAPTER 5: SUMMARY AND CONCLUSIONS ......................... 161
  5.1) THE SIMULATED AND PROTOTYPE MOBILE ROBOT .......... 161
  5.2) THE MEANS-END PATH FINDING AND FOLLOWING ALGORITHM ................................................................. 162
  5.3) THE POSITION LOCATION PROCESS ........................................... 163
  5.4) CONCLUSION ............................................................................ 164
  5.5) RECOMMENDATIONS ............................................................ 165

REFERENCES AND BIBLIOGRAPHY

PUBLICATIONS

APPENDIX A: GRID ENVIRONMENT PROGRAM LISTING .............. A1

APPENDIX B: THE QUASI-CONTINUOUS ENVIRONMENT PROGRAM LISTING ................................................. B1

APPENDIX C: THE PROTOTYPE MOBILE ROBOT .......................... C1
Chapter 1

AN INTRODUCTION TO MOBILE ROBOTS

1.1) HISTORY

With the progress of technology of all kinds, there has been a greater level of dependence and reliance on the use of mechanisms to improve man's capability to perform work. Computer control has replaced human supervision that was often dull and monotonous, and automation has reduced the delay between initialisation and execution of an idea, product, or communication. With office automation for example, fewer people were able to organise more communication of data within an organisation as more computers and advanced technology and ideas were introduced.

The flow of data in an office environment has been automated, but in an industrial production environment people still direct the flow of component parts by driving forklift trucks, etc, or by dedicated machinery such as conveyor belts. There is a need to automate the flow of parts between machining centres to take advantage of the Flexible Manufacturing Systems (FMS) now being introduced. This requires some form of flexible autonomous product carrier which has stimulated research into Automated Guided Vehicles (AGV) [AGV-1 1981, AGV-2 1983] and more particularly mobile robots.

1.1.1) What is a Mobile Robot

Robots usually conjour up visions of mechanical arms but mobile robotics were not primarily concerned with mounting a
mechanical arm on a mobile base, although this may be of interest. Mobile robotics was concerned with the passage of some product, material, etc, from one position in an environment to some other different position, without the active guidance of human controllers. Primitive versions are already in use in the form of Automated Guided Vehicles which follow a pre-laid guide path, sometimes with a limited choice of actions at junctions [Berutti 1981, Knotts 1981]. By reducing the dependence on pre-laid paths the vehicle becomes capable of self-guidance and some AGVs can now manoeuvre autonomously for short distances to accommodate areas where guide tracks are difficult to install.

Most of the vehicles required would be derived from present day forklift trucks, aeroplanes, ships, etc. The particular interest was in ground effect vehicles as the other types of transport, mainly air and sea, already had their own versions of automatic guidance [Andreussi 1978, Cremer 1978]. The passage of a forklift truck within a changing complex factory environment, was considered a problem of obstacle avoidance rather than navigation; the vehicle had to find a path if one existed, find the best path, and then follow the planned path.

1.1.2) Previous Research into Mobile Robotics

The interest in mobile robotics has increased rapidly in the last few years in several areas. One area was military where the research was directed towards mobile mines [Nelson et al 1983]. Another area was the increasing influence of the concept of automated factories and FMS [Rathmill 1983]. Two other areas generated interest: The NASA Mars Rover [Shen and Yerazanis 1979, Longendorfer 1976, Lewis and Johnston 1977] and the field of Artificial Intelligence [Bond 1978, Raphael 1975, Bullock et al 1983].
1.1.2.1) The NASA Mars Rover

The aim of this project was to produce a device suitable for landing on Mars, and then operating under its own onboard control with only general guidance from Earth. The need for considerable autonomy in the robot arose from the impossibility of direct control because of the long passage of time, approximately twenty minutes, for signals to travel from Earth to Mars and back again. This work had several interesting features such as the use of a scanning laser rangefinder [Lewis and Johnston 1977] to survey the environment and build up an internal map, which was then used to choose a route [Longendorfer 1976].

1.1.2.2) Artificial Intelligence

An alternative approach to the Mars rover was taken at the Stanford Research Institute in America, with the production of "Shakey" [Raphael 1975, Nilsson 1969, Fikes and Nilsson 1971], a mobile robot for use in Artificial Intelligence research. The main interest was in problem solving and the robot was used mainly for those purposes. A very simple environment was constructed of three square rooms connected by doors, which contained three or four simple blocks. The robot was then posed the problem of manoeuvring the blocks into certain arrangements without being instructed as to how to go about it. The problem of movement in such an environment was a simple one accomplished by an algorithm that joined two points together with a straight line. The positions of the two points were chosen by the problem solver.

Other centres used mobile robots as research tools for Artificial Intelligence studies [Bond 1978, Moravec 1979, McCalla and Schneider 1979, Coles et al 1975] or into vision research [Gennery 1979].
1.1.2.3) Automated Guided Vehicle Systems

Unmanned carts which followed pre-laid guide paths such as white lines or buried cables, have been used in industry for nearly twenty years [Knotts 1981] but it was only recently that the application of computer control was considered so that these vehicles could be used in FMS based factories [Tajima 1981, Gaillet 1981]. Although installation costs were not high and expandability was common, restrictions were placed on the cart movements which prevented their use in less constrained environments (i.e. outside factory environments).

1.1.2.4) Mobile robotics

Those areas mentioned previously were concerned with sections of other research which overlapped into the area of mobile robotics. Each area had some aspects that were of use and much was learnt from them.

Other places where research was concentrated specifically on mobile robots were the University of Warwick project [Larcombe 1978, Larcombe 1981, Larcombe 1983] which had possession of an industrial cart suitable for adaptation to unmanned use, and the LATEA Institut National des Sciences Appliques with the HILARE mobile robot for research into the navigation problem [Bauzil et al 1981, Ferrer et al 1981, Giralt et al 1978]. These later mobile robots made use of television, laser rangefinders and ultrasonic sensors to investigate the environment.

1.2) AN OUTLINE OF BASIC THEORY AND PROCESSES

Two items were of importance for a mobile robot, one being the functions within the cart, such as status, sensing, path finding, etc, and the other was the position of the cart in the environment. Both of these were required simultaneously
in real-time and pointed the way to a distributed control system for mobile robots [Moravec 1979].

1.2.1) Fundamental control concepts of a mobile robot

The following list of requirements of a general mobile robot was deduced from a study of the available literature (see References and Bibliography). The mobile robot should:–

a) be self-contained with chassis management;
b) be able to function in real-time;
c) have environment sensing;
d) be able to manipulate and update a model of the environment;
e) be able to simulate itself moving through the model environment;
f) be able to determine its own position in the real environment and relate this position to the model environment;
h) be able to find paths in the model environment and then follow them in the physical environment;
i) place as few restrictions on the environment as possible.

1.2.2) A self-contained chassis

A mobile robot is inherently a wide ranging device, and was expected to be able to move through large areas and distances relative to its own size, which range from well ordered and defined areas (e.g. stores), to shifting, irregular and temporary areas (e.g. goods delivery, or external to the factory covering). To achieve this while the vehicle was connected to a central controller either physically by wires, or by a non-contact method, for example an infra-red communication channel, was considered to be unrealistic because of the difficulties of maintaining contact. Also, being untethered the chassis had to be self-
powered, and to achieve the longest usage some form of chassis management was required. The most suitable form of power was provided by batteries, as combustion engines produced a very hostile environment for electronic devices.

As the amount of electronics on the mobile robot was increased, it was important to ensure that minimal energy was consumed, which the chassis management accomplished by switching off unused sensors and powering down processors and associated components when not required (for example loading and unloading).

Finally there was the problem of compensating for changes in the load inertia due to loading/unloading, battery discharge, etc. Changes occurred in acceleration, maximum velocity, cornering radius and halting distances, and the control system should be aware of these changes in order to compensate for their effects, for example larger masses moved at lower velocities.

1.2.3) Real-time control when moving in the environment

Because of the danger of damage to both the robot and the environment, which may contain people, then contact with any object must cause an immediate halt. There must be minimal delay between the detection of the object and the stopping of the chassis and as deceleration from a fixed velocity to zero cannot be achieved in zero time, the maximum velocity was limited.

However, the maximum operating speed was required to obtain maximum efficiency and there was a requirement that all processes must not slow down the movements more than necessary (i.e operate in real-time).
1.2.4) Environment Sensors

To be independent the mobile robot required some method of sensing the environment so that obstacles were detected and avoided and routes planned and checked. The available methods of sensing are divided into three categories.

The most obvious one is television, an area where large amounts of research effort have been directed [Moravec 1979, Gennery 1979, Ferrer 1981, Luh and Yam 1981, Jain et al 1979]. The images from television cameras were processed and deductions from the resulting data made concerning the arrangement of objects in the environment.

The second category of sensing was the tactile bump sensor which had the advantage of being simple to operate, but the disadvantage of a short range, although a successful implementation was possible if the environment was kept very restricted [Marce et al 1980, Winston 1975]. A tactile sensor was unsuitable in an unknown or constantly changing environment because of the short range. The short range was put to good use to provide an emergency sensing of objects close to the mobile robot, or for moving in restricted spaces.

The final category of sensing was distance ranging devices which had the advantage of providing accurate distances to objects which was something not easily obtained from television. One disadvantage was that the amount of information obtainable at any one time was small, which conflicted with the operational requirement that the environment should be scanned quickly. Of the three main types of distance ranging, laser, ultrasonic, and electromagnetic only the laser rangefinder has had any significant success as a short range (i.e. less than 10 metres) environment scanning device [Lewis and Johnston 1977] due mainly to the introduction of laser diodes.
Radar was a well researched area but little effort had gone into producing a scanning radar for short range operation. Ultrasonic devices were difficult to scan due to the long time constants involved between transmitted and received signals. The normal approach was to use multiple ultrasonic ranging devices with fixed directions [Larcombe 1978, Bauzil 1981].

1.2.5) Modelling the Environment

The control of a mobile robot was basically an abstract concept where the combinations of stimulations caused by the environment resulted in a decision being made which was then translated into actions in the environment. This process could be considered as a subset of the subject termed "Artificial Intelligence".

Some form of interpretation or translation was required between the real world and the abstract decision process, and this was provided by the environment model used. The model provided a suitable format in which plans and actions were formulated and sometimes simulated, to improve the probability of success of the same actions in the real world. The translation from model to real environment and vice versa was accomplished by the control and sensing process.

The process depended on a coherent and non-contradictory set of rules determining the model make up, and how these models were related to the real world. For example, modelling all objects as squares required a set of rules to cope with round objects, both in the model and the real environments.

There were two basic types of models that were used in mobile robotics, and also robotics in general and these were firstly the temporary model, obtained only from the environment sensors at a particular moment in time and
space, and secondly the permanent model stored in memory which contained previously obtained environment information.

In the temporary model very little previous knowledge about the environment was retained in memory, and the model was reconstructed whenever required. The real environment was sensed in some way and the data available used to construct a model of the immediate surroundings which was valid only for that particular location [Moravec 1979, Nakamura 1981, Cahn and Phillips 1975]. At every new location and time a new model was formed.

The alternative was to store a complete map of the environment to allow path planning techniques to determine the optimal manoeuvres. Temporary modelling tended to be a literal translation from sensor data into models as the result was not permanent, and a variety of models were satisfactory. For permanent maps an efficient modelling process was required because the available storage space for the model was never sufficient for detailed representations, and a compromise was necessary between models which required less memory space per unit area which enabled a larger environment area to be active, and a more detailed model with a smaller active area. Path finding was a strategic (i.e. global) process and a model which actively represented the largest area of the environment was preferred. Less detailed models relied on symbolic descriptions and required greater processing resources to enable them to produce useful actions.

Initially the mobile robot was supplied with a ready-made environment representation in much the same way as buying a road map from a shop. Alternatively the mobile robot was allowed to construct such a map as it manoeuvred, starting with an initially blank or limited model, i.e. walls, pillars, etc. This process must maintain an accurate relationship with all the other objects which had been
modelled already, and required that the location of the robot be accurately known when updating was taking place. This illustrated the idea of checking the model against the real environment as the robot was manoeuvring, and updating the model whenever they disagreed. Both types, temporary and permanent, were considered necessary for different aspects of the control mechanism (for example temporary for position finding (see section 2.3) and permanent for pathfinding (see section 2.2)).

1.2.6) Simulating the mobile robot

Having simulated the environment it was necessary to simulate the movement of the mobile robot in the environment to obtain the most accurate planning of actions possible. For the temporary model the simulation was required to check that the physical action of the robot matched the predicted action to maintain a high probability of success. Should the physical action not be an accurate execution of the modelled action this could cause collisions with obstacles. However as each new action generally had no reliance on the previous states of the mobile robot and only the accuracy of the initial state was important, the ability to simulate itself was not considered a necessary requirement.

For the permanent environment model an accurate simulation of the mobile robot was important, as the starting point of each new action was dependent on all previous actions. Also, without simulating itself it was difficult to plan actions with a high probability of success, because the physical properties of the mobile robot e.g. width, turning circle, etc. affected the set of possible actions. A plan which included an action such as manoeuvring through a gap narrower than the mobile robot was certain to fail and must be detected. Certain properties of the robot must be accurately simulated for correct operation of the planning techniques. For mobile robots these properties were orientation, position and size.
Some environment representations can be arranged so that references to the physical quantities of the robot were limited. For example, in a grid environment if the grid square size were larger than the area occupied by the robot, then any path found linking grids by Moore's algorithm [Sutherland 1969] would be a practical route. This assumed that physical quantities such as turning radius, speed and stopping distances, were not important as they fitted the existing environment representation. This type of environment suffered from drawbacks as the accuracy of the environment model was reduced as the grid size was increased and viable paths were sometimes obscured. This showed that the environment model was interrelated with the model of the mobile robot itself, and that a consideration of both models was appropriate.

1.2.7) Determination of the position of the mobile robot

There were two main reasons why the position of the mobile robot needed to be known. The first was so that the mobile robot detected when the specified destination was reached, in order to terminate correctly the path following manoeuvres. The second reason was more relevant to the permanent environment model system. By determining the position of the mobile robot, the difference from the predicted position was calculated and used to correct any errors. This process was effectively closing the feedback loop so that deviations from the chosen route were detected and corrected. The sampling rate of the position finding process was important for the stability of this process; too slow and the errors become very large, too quick and resources were wasted.

Assuming that the position was known accurately initially there were several techniques [Bauzil et al 1981, Tsuji and Kawashima 1981, Lezniak et al 1977] for determining the change of position with respect to time. These were used to
update, or extrapolate, the present position of the mobile robot between absolute position fixes.

1.2.8) Path finding and following

As already discussed, the environment and mobile robot representations were interrelated and were expected to effect the route finding and following algorithms. Hence as there were two types of environment model, temporary and permanent, there were two types of path finding algorithms.

One type used only local conditions to decide which movement was likely to lead to a decrement in the distance separating the mobile robot from the destination. This decision was usually based on a set of heuristic rules about the environment, describing the characteristics of the obstacles, walls, corners, etc, which allowed actions with a high probability of success to be formulated in local environments. Because of the lack of previous data used in the representation, the path finding and decision making process was simple and easy to implement. The disadvantage was that if an environment did not conform totally to the known set of rules, the anomalies could cause the decision making process to manoeuvre the robot into a situation from which it was unable to move. Also, because only local knowledge was used, the paths found were not always the best available.

Path following was usually irrelevant in these circumstances as there was no memory of where the robot had been and checks to determine the accuracy of the actions performed were unnecessary. Each decision point was considered as a new environment although usually some measures were taken to ensure that the robot did not oscillate backwards and forwards between two situations.
The alternative was to use a permanent model of the environment which allowed a variety of different techniques of path planning to be attempted. The basic principle was to select a number of routes through the environment based on the arrangement of free space and obstacles, from the infinite number of possible paths which joined any two points. This was achieved by the path finding part of the control. These paths were then assessed taking into account the various advantages and disadvantages. The optimal path was then selected to be used in the path following part of the control.

1.3) APPLICATION AREAS OF THE RESEARCH

The field of mobile robotics depended upon the interaction of many disciplines and it was expected that many of them would benefit from this type of research. The areas listed below were restricted to those which would benefit from research into algorithms for controlling a mobile robot in an environment using a scanning range finder.

1.3.1) Mobile robotics

The main area expected to benefit from this research was the field of mobile robotics itself. That there has been a need for mobile robots was demonstrated by the interest in AGV's [AGV-1 1981, Larcombe 1979] for use in industrial environments. These vehicles which relied on pre-laid guide paths were the fore-runners of untethered roving vehicles (i.e. mobile robots). Also there was considerable interest in Flexible Manufacturing Systems for automating the industrial manufacturing processes, and an inherent part of such systems was the movement of material during the various manufacturing processes [Seals 1984a]. This was normally achieved with manual intervention but new factories and systems were being designed with AGV's fully supported [Tajima 1981, Stoize 1981, Koenig 1981] which were only one step away from installing mobile robots.
1.3.2) Robotics in general

Other areas in robotics have benefited from this type of research, and one such area has been in providing obstacle avoidance methods for robot arms working in cluttered spaces [Kuntze and Schill 1982, Powell 1982, Petrov and Sirotka 1981] or in workspaces used by more than one robot.

1.3.3) Transport (air, land and sea)

Attempts to automate the process of manned road vehicles whether it be the vehicles themselves [Darenberg 1981, Norton-Wayne and Guentin 1981] or the complete road traffic system [Fenton et al 1978, Leininger 1978, Cremer 1978] have been made and London Transport has introduced a method of determining the position of buses by the use of inductive loops buried in the road as 'sign-posts' [Clarke 1980]. The bus then transmits the position to a central controller where decisions are made on how to obtain the best performance.

In shipping where the environment is less restricted as anywhere the sea is deep enough the ship can travel, autopilots and extensive automated controls have been installed, particularly in recent years [Lambalieu 1976]. There is also the advantage of being able to determine exactly where the ships are because of the navigation satellites in use [Mara 1973, Stansell 1973, Fjelleheim and Gjeruldsen 1973] which act as beacons transmitting position information to the ships. They also enable the position of other ships to be known [Karmarker et al 1976, Ohrstrom 1973]. The accuracy of 50 metres was satisfactory for a ship which is travelling hundreds or thousands of Kilometres.

Rail transport is a much more restricted environment as it was only possible to travel on the rails, and generally only
forwards. This restriction allows the use of an automated form of transport with trains controlled by computer without guard or driver.

A system similar in many ways to sea traffic control, is air traffic control which controls the movement of aeroplanes in a three dimensional space. Only by stipulating imaginary air routes and regulations is it possible to maintain an orderly system supervised by air traffic controllers. Any device which plans paths and detects deviations from planned paths automatically would be advantageous, and there may be a contribution in that area.

There is one large drawback with all these forms of traffic, and that is the safety of the passengers. This can not be guaranteed with robot control, nor can it with human control, but the ability to make decisions in a crisis is something that would be difficult to program into a robot controller, even though overall travel might be safer. However these forms of vehicle controllers may find acceptance in the transportation of goods rather than people should the economics prove viable e.g. automated mail carriers.

1.3.4) Environment sensing

So far most environment sensing for any robot has been achieved by the use of television cameras, which had the range and flexibility to cope with almost any environment and situation, but also the difficulty of extracting the necessary data from the large amounts available.

Rangefinders provide the required obstacle data but with the difficulty of organising the data so that the important sections of the environment were "pictured" as a whole. It was hoped that this research would contribute towards the range data processing algorithms which performed these functions.
1.4) THE OBJECTIVES OF THE RESEARCH

The aim of this research was to determine the validity of the requirements proposed as the fundamental control concepts of a mobile robot in section 1.2.1. Work was centred on the three main topics, outlined below. The remaining aspects, i.e. self-contained chassis and real-time operation were treated as secondary objectives to be demonstrated if possible.

1.4.1) The path finding and following techniques

The main purpose of any mobile robotic vehicle was to manoeuvre from one location to another, normally within the same room but occasionally moving into a different room. To achieve this some method of path finding in a simulated environment suitable for all possible situations was required with the ability to follow the planned path in the physical environment.

1.4.2) The construction of environment models from sensor data

The data from the sensors was to be used to construct a model environment suitable for use by the path planning techniques.

1.4.3) The determination of the position of the mobile robot

The ability to follow a planned path and to update the environment model from the data obtained from the sensors, required that the position of the mobile robot was known accurately.
1.5 SUMMARY OF THE INTRODUCTION

This chapter considered the fundamental requirements of a mobile robot: that it must be self-contained and operate under real-time constraints, that environment sensing and modelling with the ability to simulate the mobile robot manoeuvring in the model environment and techniques of finding and following paths were required, and that position finding methods were necessary for correct operation.

The thesis was described in section 1.2.1 and the objectives of the research which were to be verified were outlined.
CHAPTER TWO

AN INVESTIGATION OF MOBILE ROBOTICS

This chapter considers in detail the main ideas of the thesis indicating how the conclusions were influenced by previous research. Aspects deduced in chapter one were investigated, previous implementations of mobile robots considered and conclusions drawn as to a suitable implementation for a prototype mobile robot. A limit on man hours available meant that only those areas necessary for the investigation of the main ideas of the thesis (see section 1.2.1) were implemented in detail. In other areas, simplified techniques were used which resulted in a limit being set early on, on the amount of memory and processing resources available. Additionally a simple mechanical structure was adopted for the construction of the chassis of the mobile robot. Four areas were considered: environment modelling, path finding, position location and environment sensing.

After extensive reading (see References and Bibliography) a conclusion was reached that controlling the mobile robot by "artificial intelligence" techniques was a possibility, but that the resources required were beyond those available. Therefore no further references were made to that subject, although it was noted that if more resources were made available then these techniques would need to be re-investigated.

2.1) MODELLING THE ENVIRONMENT

To enable the mobile robot to act in an intelligent manner a model of the environment was necessary to determine what actions needed to be taken before the necessity of executing
them was apparent. The model was required to include previous knowledge of the environment and to have a consistent set of rules for interpretation (e.g. objects, free space) and translation into actions in the physical environment. Additionally there were to be facilities for extending and altering the model on the basis of information received from the environment sensors.

In section 1.2.5 two types of environment model, temporary and permanent, were outlined and it was considered advantageous for the mobile robot if both could be used without significant changes in the control techniques.

A restriction was that only two dimensional models using the X-Y cartesian coordinate system were considered. This was because the majority of mobile robots operated on flat floors. Changes in height caused by moving to a different floor were to be accommodated as a change to a new environment.

2.1.1) The grid environment

One approach divided the robot's environment into squares which either contained an obstacle or empty space [Eastman 1970] (see figure 2.1). The problem was then one of finding a path through a maze. A suitable minimum square size of one tenth the robot area was a useful compromise between accuracy and memory requirements. For example, assuming an environment of 100m x 500m, which was small for an industrial setting, there was a total area of 50,000 square metres, and with a robot of 1m x 3m (= 3 square metres) there were 166,666 different memory locations required to retain such a map, the majority of which would be empty. This type of environment allowed standard maze solving algorithms to be used [Sutherland 1969] but required a large memory, and was still only a crude approximation to the real
Figure 2.1: A simple grid environment containing one object
world. It did however have the advantage of being extremely simple to manipulate, as objects were entered or deleted by changing the status of the appropriate memory locations (e.g. object, robot, empty).

2.1.2) Listing the objects and/or the empty spaces

An alternative model made use of the large redundancy inherent in direct mapping, by either listing the obstacles (size, shape, location) or listing the free areas (size, shape and location) [Giralt et al 1979]. In this way only a small amount of memory was required to describe an environment arrangement (see figure 2.2(a)).

Listing the objects was easy to achieve as each object was defined explicitly by the perimeter shape but methods of finding routes through such environments were not simple [Lozano-Perez 1983, Brooks 1982, Lozano-Perez and Wesley 1979]. However, listing the spaces was complex because the areas were not completely defined and imaginary boundaries were required to complete the segmentation. The choice of boundaries was somewhat arbitrary and could lead to routes becoming obscured. This listing was itself more complex (see figure 2.2(b)) but finding a path was simpler, and was achieved by connecting adjacent empty spaces. For example, in figure 2.2(b) spaces \( l_1, l_5, \) and \( l_3 \) gave a path from left to right. Obviously some way of defining the position within each space and the crossing point between the spaces was also necessary.

To add to the list of objects the object had to be surveyed from all sides [Kauffman 1983] and the exact shape and position determined. Removal was achieved by surveying the area where the object was mapped to ensure that no part of it remained, and then deleting from the list.
Object list
1) B,C,D,E,A
2) G,H,I,J
3) K,L,M,N

Space listing
1) A,F,N,M,R,O
2) A,B,C,D,J,G,P,O
3) G,H,I,K,L,Q,P
4) L,M,R,Q
5) D,E,F,N,K,L,J

Figure 2.2(a): A model of the environment listing the objects

Figure 2.2(b): A model of the environment listing the spaces
For the list of spaces the surveying operation was the same, except that every time an object was added or deleted the spaces required redefining.

Listing the objects was considered to be the preferred technique due to the ease with which an environment could be described.

2.1.2.1) Representation of objects as polygons or circles

The objects were represented by straight line approximations of the perimeters which could require many straight line segments (see figure 2.3). Alternatively, convex polygons could be used. A convex polygon is a many sided shape where the external angle of every vertex is greater than $180^\circ$ so that eventually an enclosed shape is formed (see figure 2.4). This resulted in a simpler model with less sides.

The disadvantage was that the error of the model was greater as the object must reside completely within the model boundary to prevent collisions from passing robots. Greater accuracy was obtained by using two or more smaller overlapping convex polygons to model the object. [Lozano-Perez and Wesley 1979].

If required, further simplification was possible by modelling the objects by circles which only required a centre point and a radius to describe them completely. This increased the error of representation considerably as the circle diameter had to be at least as great as the widest part of the object (see figure 2.5). The environment then consisted of a series of circles of varying radius and location. Greater accuracy was obtained by modelling the object as a set of smaller overlapping circles, similar to the method used for convex polygons.
Figure 2.3: Polygon model of an object perimeter

Figure 2.4: Convex polygon model of an object perimeter
Figure 2.5: Circular model of an object perimeter

Figure 2.6(a) The environment model

Figure 2.6(b) The environment list
Due to the inherent inaccuracy of modelling the environment using circles the convex polygon method was preferred. This allowed the accuracy of representation to be varied to determine the effect on the other processes of the mobile robot.

**2.1.3) The mobile robot as a charged particle**

A different type of environment model which did not attempt to directly map the environment was the electrostatic model [Kuntze and Schill 1982, Shih 1982]. The principle was that the mobile robot and all objects contained a charge of the same polarity and the destination contained a charge of the opposite polarity. In this way the mobile robot was repelled from the stationary objects and attracted towards the destination. The force on the mobile robot caused by all the objects summed to a single vector as did the attraction of the destination. The resultant of these two vectors determined the path the mobile robot followed.

Alterating the value of the charges on the objects could be used to alter the characteristics of the environment. A destination with a large attracting charge straightened out the route, and a large opposing charge on an object forced a wider passing distance.

For a particular environment there were a limited number of "valleys" between objects which simplified the pathfinding problem. This had similarities with the paths produced when the free space in the environment was modelled by generalised cones (see section 2.2.1.4).

This environment has been implemented on a mobile robot by Larcombe at the University of Warwick.
2.1.4) Conclusion

From a consideration of the environment models and the path finding models (see section 2.3) it was concluded that the type of model to be used was determined by the path finding technique chosen. Therefore to maintain a general technique the listing of closed convex polygon models of objects in the environment was selected. Variations in the implementation were possible as shown by the grid (chapter three) and quasi-continuous environments (chapter four) implemented.

2.1.5) Proposed technique of modelling the environment

From the conclusions drawn after a consideration of the various alternatives, the following model of the environment was proposed which was to be implemented for the simulated and prototype mobile robots.

The environment was to consist of a list of (closed convex) polygons which modelled the perimeters of the objects. Normally the polygons were isolated in the environment but if required two or more could be placed adjacent or overlapping in order to obtain more complex shapes. The first implementation was to use the grid coordinate system (see section 3.2.1) which had an X-Y orthogonal axes coordinate system with positive integers only. The list of objects consisted of rectangles aligned with either the X or Y axis, denoted by the position of the bottom left-hand corner, plus the length of the rectangle in the positive X direction, and the width in the positive Y direction. The listing of a sample environment (see figure 2.6(a)) was shown in figure 2.6(b). Diagonal or other shapes were obtained by placing rectangles in adjacent grid squares.

2.2) PATH SELECTION AND FOLLOWING

The main criteria for selection was that it should be possible to implement on the prototype and simulated mobile
robots with restricted memory and processing resources, and have the potential to operate in real-time or near real-time conditions.

There were two modes of operation required; the first was that the mobile robot should be capable of simulating movements and actions in the model environment in order to plan paths (see section 2.2.1), and secondly to operate satisfactorily in the physical environment without reference to the model environment, for situations where the model environment was found to be inaccurate. This allowed either a permanent or temporary model of the environment to be used, as discussed in previous sections.

Having planned a path the mobile robot must be capable of following it (see section 2.2.2) with the desired degree of accuracy.

2.2.1) Planning the path using the simulated environment

If the complete environment was known then standard maze solving techniques and algorithms could be used. These are discussed below with their advantages and disadvantages. Additionally, provided that there was at least one continuously known section of the environment joining the start and finish locations, it was possible to find a (possibly sub-optimal) path through a partially known environment. This was the more general solution and the one in which the mobile robot generally operated, as a perfectly known environment did not require a mobile robot as paths would be pre-planned. This class of problems was called "spatial planning" problems [Lozano-Perez 1983].

2.2.1.1) Planning the path through passageway mazes

Objects in a grid environment form short passageways and paths connecting these were found by using Moore's algorithm
[Sutherland 1969]. This algorithm placed a number one in all the squares adjacent to the starting location square. The number was then incremented and placed into all adjacent squares containing a lower number. This process was repeated until the destination square was reached (see figure 2.7). The shortest paths were found by moving from the destination to the start by always moving to a lower numbered square. In this example nine steps were required and there were two possible routes, one of which had many different ways of being transversed, but each route had the same length. This method depended on the environment being known before travelling through it and did not work for unknown environments.

2.2.1.2) Finding paths when the environment was modelled by circles

If the grid environment was not used but instead objects were modelled by circles [Moravec 1979] a different algorithm was used to solve the path finding problem. The problem was one of manoeuvring the robot, which was also represented by a circle, through an environment consisting of many circles, without any overlaps occurring between object circles and the robot circle. The problem was simplified by adding the radius of the robot circle to the radius of all object circles, and then shrinking the robot to a one dimensional point. Any path which avoided all intersections with circles would be a valid one.

The shortest path was found by constructing a data structure containing all possible paths, and then by using a shortest path algorithm [Fikes and Nilsson 1971] to sort through the possible paths to find the optimal one. The paths in the data structure were found by knowing that there were only four different paths between two circles, two outer routes, and two inner routes.
Figure 2.7 Path finding using Moore's algorithm

Figure 2.8 A general truncated cone
This structure had several advantages; it was easy to construct paths through the environment and to make allowances for the physical size of the mobile robot. The disadvantages were judged to be the inaccuracy of representing objects by circles as large areas of free space were obscured by this technique, and that the environment must be known before the path planning commenced.

2.2.1.3) Finding paths when the environment was modelled by polygons

Instead of modelling the objects as circles a more accurate representation modelled them as polygons [Lozano-Perez 1983, Kuntze and Schill 1982, Lozano-Perez 1981, Lozano-Perez and Wesley 1979, Nilsson 1969], as described previously. The process minimised the number of vertices required to enclose a space greater than or equal to the real object. The model mobile robot was shrunk to a one dimensional point and the appropriate amount "grown" onto all the objects.

Valid paths were constructed and the shortest found by the appropriate algorithm. Paths consisted of straight line sections joining vertices of "grown" polygons connected by stationary rotations.

2.2.1.4) Path finding in a polygon model of the environment using truncated cones

A truncated cone was an area whose perimeter was defined by the facing sides of two objects, assuming that the objects were modelled by polygons. These defined the changing radii section of the cone and linear extensions from the maximum and minimum radii defined the complete general cone (see figure 2.8).

The linear sections were extended until they overlapped with those of another cone. The spine was found by halving the
angle formed by the converging sides, and extending it until an object was encountered. A set of main paths and avenues was formed by the intersecting spines and as only one spine or path was possible between two objects, the path finding process was simpler [Brooks 1982].

Any paths found by this method were not necessarily the shortest, but they had several advantages:

a) For a sparse distribution of objects the main routes in an environment were identified quickly.

b) The paths found were at a maximum distance from the objects at all times which allowed the largest variety in the shape of the mobile robot.

c) No reference was made to the coordinate system being used.

This technique had disadvantages:

a) The paths were not necessarily optimal as shortcuts between spines may exist which were not discovered.

b) For complex environments the construction of suitable generalised cones was impeded, restricting the cone size which increased the complexity of the routes.

c) The environment must be completely known to find a path.

2.2.1.5) The Means-end algorithm for finding paths

Several other mobile robots used a general Means-end algorithm outlined below:

a) If the robot is at the destination then stop.
b) Taking into account all possible restrictions select the manoeuvre which will move the mobile robot such that the distance to the destination is reduced by the greatest amount.

c) Repeat from (a).

This algorithm was suitable for operation in completely unknown environments provided that sensor data on the immediate surroundings was available. This type of situation was similar to that of the Mars Rover [Shen and Yerazamis 1979, Lewis and Johnston 1977] which used a simple two dimensional computer simulation that incorporated a scanning laser rangefinder to discover objects, and a Means-end algorithm for manoeuvring [Longendorfer 1976].

A mobile robot was developed in 1980 in France at the Laboratoire d'Applications des Techniques Electroniques Avancees, Institut National des Sciences Appliques, Rennes, by Marce [Marce et al 1981a, Marce et al 1981b] which initially used tactile sensing but later added optical sensing. This used a similar Means-end algorithm to negotiate objects and straight line paths when no objects were detected.

Other research into mobile robots and path selection was performed [Moravec 1979, Cahn and Phillips 1975] which achieved a wider perspective in the use of this algorithm.

The disadvantages were that it could not be guaranteed that if a path existed in an environment it would be found, or that any paths found would be optimal.

2.2.1.6) Conclusion

Existing methods of path finding apart from the Means-end algorithm consisted of techniques which were successful only for completely known environments. Additionally they
required extensive computing resources. Both these requirements were outside the specification outlined initially and the conclusion was that they were not suitable for this research project. Therefore the Means-end algorithm was taken as the starting point for the path finding process.

2.2.1.7) The proposed path finding technique

After a consideration of existing path finding techniques three useful themes were observed which were used to propose a suitable method. The first theme was that the closed polygon model of the environment was found useful in the majority of techniques, secondly that the majority of path finding methods relied on linking specific points in the environment into a coherent structure which was then interrogated to determine possible solutions, and finally that the Means-end algorithm functioned correctly in all types of environment, from known to unknown. An algorithm based on the Means-end algorithm was proposed which used a simulated environment, as given below:

a) Form a linked list of positions to be visited.

b) Orientate with the destination and move towards it unless obstructed.

c) If obstructed choose an orientation which avoids the object and continue moving until the obstruction is cleared.

d) Add the position of the mobile robot (the Intermediate Turning Point) into the linked list of destinations.

e) Repeat from (b) until the destination is reached.
f) Return the mobile robot to the start position and using the updated list containing the added Intermediate Turning Points repeat from (b) until no objects are encountered at any point on the path.

The path found in the simulated environment was then free of collisions and was suitable for execution by the prototype mobile robot.

This algorithm apart from part (f) could also be executed by the prototype mobile robot as it was manoeuvring through the environment.

For a simulated mobile robot the environment data came from simulated sensors whereas for the prototype mobile robot the data came from the sensors scanning the immediate surroundings.

If a selection of paths was required they were produced by altering the characteristics of part (b) of the algorithm which selected the new directions so that alternative decisions were used.

2.2.2) Path following

Having found the path, the problem became one of following the chosen path [Tsumura et al 1982]. There were two important properties to be considered; the deviation of the mobile robot from the planned path (see section 2.2.2.1) and the effect of the physical characteristics of the mobile robot on the path (see section 2.2.2.2). A conclusion was drawn (see section 2.2.2.3) as to the necessity of accurate path following methods.
2.2.2.1) The deviation of the mobile robot from the planned path

The position of the mobile robot was determined and the difference from the planned path calculated and some action taken. For example:

a) If the error was decreasing slowly then do nothing.

b) If decreasing quickly, reduce gradient of the decrease, by steering away from path.

c) If increasing slowly, make a small change of direction towards the path.

d) If increasing quickly attempt to reduce the rate of the increase until the process in (c) can be used.

Apart from attempting to follow the planned path there was the possibility of unknown objects causing a deviation of the mobile robot from the path (see figure 2.9).

A decision must be made whether to backtrack to some point, $Y_1$, where the object no longer presented an obstruction, or to use some local path addition to join the route at some point further along, such as $Y_2$ or $Y_3$. This required the use of the path planning algorithm.

2.2.2.2) The effect of the mobile robot's physical characteristics on the path following problem

When the path was planned little reference was made as to how the physical characteristics of the mobile robot, apart from size, affected the path. These effects must be allowed for in the path following. For example, when turning corners a compromise was necessary between velocity and turning radius, for faster velocities a larger turning radius was
Figure 2.9 Rejoining the planned path after a diversion
required. As the turning radius was specified by the path finder, then the velocity was controlled by the path follower to ensure the limits were not exceeded.

If more than one route was possible then a consideration of physical mobile robot characteristics could be used to produce a weighting factor, so that a longer route with less turns was more desirable for a mobile robot with a large mass and restricted braking, than a shorter but more twisting route.

2.2.2.3) Conclusion

It was shown that following a previously chosen route had several complicating factors; the difficulty of returning to the chosen path once a deviation occurred and the effect of the characteristics of the mobile robot when attempting to stay on the path. After a careful consideration it was concluded that accurately following a pre-chosen path produced no significant advantage, however it was still necessary to reach the destination, and to know when the destination had been reached.

Therefore the path following process may be reduced to being able to orientate and move towards the required Intermediate Destinations (see section 2.3). With this method the deviation from the planned path was of no consequence as the mobile robot re-orientated whenever required. The important factor was the list of specified destinations and the straight line segments joining them were of minor significance.

This method of path following was compatible with the path finding algorithm outlined in section 2.1, which found the Intermediate Turning Points when avoiding objects. By constructing a linked list of these points the path was defined by the straight line segments between successive
When following the pre-planned path any deviation from the straight line was unimportant as there was re-orientation at a future point. Only the frequency of re-alignment needed to be decided and this was related to determining the position of the mobile robot (see section 2.3).

Also if unknown objects blocked a pre-planned path a local diversion around it was found by the original path finding algorithm (see section 2.1) which joined up with the next Intermediate Turning Point, and this presented no problems to the path finding technique.

2.3) LOCATING THE POSITION OF THE MOBILE ROBOT

One of the fundamental requirements of a mobile robot was the ability to model accurately its own position in the environment. A free roving mobile robot which was not physically tethered to the frame of reference was able, by identifying known features, to calculate its own position. This process consisted of three functions; recognising a sufficient number of known features, determining the position of these features, and finally calculating the position of the mobile robot in the frame of reference. Various methods of implementation were available and each was assessed for its suitability for the prototype mobile robot and a conclusion drawn indicating the optimum technique.

2.3.1) Recognition of features in the environment

There were two main types of feature recognition; using rangefinders (see section 2.4.1) which detected the physical presence of features or using television cameras (see section 2.4.2) which detected features symbolically.
2.3.1.1) Feature detection using rangefinders

The type of rangefinder was not important as tactile, optical and sonic devices all provided the same information albeit with different characteristics. A set of adjacent range measurements was required and was usually obtained by scanning the rangefinder over the environment. The information was then processed to identify the features. In a blocks world the detected edges [Seals 1984b, Nakamura 1981] were used to calculate the position of the features. An edge was detected by the sudden change in range between two successive measurements (section 4.1 discusses this process in more detail).

An alternative to rangefinders was to use beacons placed in the environment as identifiable features. Two types of beacons were developed for mobile robots; passive, which were detected from the robot (e.g. by infra-red light) [Bauzil et al 1981], or active transmitters (e.g. electromagnetic) [Lezniak et al 1977]. By a careful selection of the beacon types and positions in the environment they can be identified. Three or more beacons needed to be detected to obtain an unambiguous solution to the position calculations.

The multiple beacon method removed the majority of the feature identification process which was an advantage because it was resource expensive, but had the disadvantage that beacons must be placed in the environment, contrary to the initial specification that the prototype mobile robot should be able to operate in unrestricted environments. Additionally, ensuring that three beacons were always detected was a difficult requirement to fulfil due to objects obscuring beacons at various points in the environment.
2.3.1.2) Symbolic feature detection

A television picture contains variations in light and dark (or colours) which are related to the environment and an object appears as a pattern of light and dark. The variations in the size and proportion of the dark and light areas were used to estimate size and range while stereo vision improved the accuracy of the rangefinding.

Stereo images allowed the range to be calculated provided that the features of interest were identified in both images [Moravec 1983, Gennery 1882]. This was regarded as symbolic identification because abstract concepts such as "a square block" were used to describe the objects. This contrasted with the rangefinder technique which only identified that "some object of an apparent size" was at a known range.

The drawback of the television sensor was that the amount of data was large and required extensive computing resources to manipulate it. Either a "simple" computer took a long time (up to 5 seconds for a single "picture") or a complex processor (i.e. array processors, parallel processing) took a shorter time (e.g. 0.5 seconds). Also the images observed were misleading as the process had difficulty in distinguishing between objects and the shadows they produced. The amount of shadow was influenced by the level of illumination which was not under the control of the data processor.

2.3.1.3) Conclusions

The use of beacons in the environment was rejected because the specifications of the prototype mobile robot required no special treatment of the environment. Television cameras were considered to be too slow and too complex for this particular mobile robot. Therefore it was concluded that the feature detection process would consist of processing the data from a scanning rangefinder.
2.3.1.4) The proposed method of detecting features

The rangefinder was to be rotated through 180° taking range measurements at fixed intervals. Isolated obstacles, which were the features required, were detected by differentiating successive range measurements and then comparing with a threshold value to detect feature edges. The sign of the edge (positive or negative) was obtained from the differentiated values and a feature consisted of a negative and positive edge pair (see figure 2.10). The position of the feature relative to the mobile robot was then calculated.

2.3.2) The proposed method of identifying the position of known features in the environment

From a study of the available literature it was concluded that methods of recognising features from the information obtained from the rangefinder had not been investigated in a format suitable for the prototype mobile robot. Therefore the following method of determining the absolute position of the features was proposed.

Methods of recognition were studied and the conclusion drawn that the features were identifiable by matching some characteristic of the objects, with a list of all the existing characteristics. It was proposed that the distances between three objects \((D_{12}, D_{13}, D_{23})\) would be used as the characteristic to be matched. The values of \(D_{12}, D_{13}, D_{23}\) were calculated from the information obtained from the scanning rangefinder using the cosine rule.

The matching process compared one of the distances (e.g. \(D_{12}\)) with the set of distances between all known feature pairs. Those within an fixed error (e.g. 10%), were grouped into a subset. Then the distance between each member of the subset
Figure 2.10(a) The environment

Figure 2.10(b) The range values obtained
Figure 2.10(c) The range measurements plotted against scan angle

Figure 2.10(d) A plot of differentiated range measurements
Figure 2.11 Identification of object triplets from the inter-object distances

Figure 2.12 Calculation of the position of the mobile robot
and all the other known features was calculated and compared with one of the remaining values (e.g. \( D_{13} \)), to form a second smaller subset of triplets of known features. The remaining value (i.e. \( D_{23} \)), was used to confirm the selection of the triplet.

Three states of the final subset were of interest: firstly, when it was empty and there were no matching triplets indicating that either there were errors in the calculations or that one or more of the detected features was not in the set of known features, secondly, the set contained only one triplet which was assumed to be a correct identification; and thirdly that it contained several triplets one of which was assumed to be the correct one.

If there were several triplet candidates the ambiguity could be resolved by calculating the positions of the mobile robot (using the process outlined in section 2.3.3) and then moving a fixed amount and repeating the position calculation process. If there were still several candidate positions these were compared with those from the previous position, noting that the position should have changed by the same amount that the mobile robot had moved. This process was repeated until the desired level of confidence in the result was achieved.

2.3.3) The proposed method of calculating the position of the mobile robot

The method used to calculate the position of the mobile robot was based on trigonometric relationships of three known feature positions. A general solution is shown in figure 2.12 where \( X_R \) and \( Y_R \) are the unknown values of the mobile robot position.

Symbols used

\[(X_1, Y_1) \quad - \quad \text{position of feature 1.}\]
\((X_2, Y_2)\) - position of feature 2.
\((X_3, Y_3)\) - position of feature 3.
\(R_R\) - distance of the mobile robot from the origin.
\(R_{01}\) - distance of feature 1 from the origin.
\(R_{02}\) - distance of feature 2 from the origin.
\(R_{03}\) - distance of feature 3 from the origin.
\(R_1\) - distance of feature 1 from the mobile robot.
\(R_2\) - distance of feature 2 from the mobile robot.
\(R_3\) - distance of feature 3 from the mobile robot.
\(\theta_{12}\) - angle between features 1 and 2.
\(\theta_{23}\) - angle between features 2 and 3.
\(\theta_{31}\) - angle between features 3 and 1.

The following equations then apply to this orientation:

\[
(X_R - X_1)^2 + (Y_R - Y_1)^2 = R_1^2 \quad \text{(1)}
\]
\[
(X_R - X_2)^2 + (Y_R - Y_2)^2 = R_2^2 \quad \text{(2)}
\]
\[
(X_R - X_3)^2 + (Y_R - Y_3)^2 = R_3^2 \quad \text{(3)}
\]
\[
X_R^2 + Y_R^2 = R_R^2 \quad \text{(4)}
\]
\[
R_{01}^2 = X_1^2 + Y_1^2 \quad \text{(5)}
\]
\[
R_{02}^2 = X_2^2 + Y_2^2 \quad \text{(6)}
\]
\[
R_{03}^2 = X_3^2 + Y_3^2 \quad \text{(7)}
\]

Multiply out (1), (2) and (3) and substitute (4), (5), (6) and (7) where appropriate.

\[
R_R^2 + R_{01}^2 - (2X_1X_R + 2Y_1Y_R) = R_1^2 \quad \text{(8)}
\]
\[
R_R^2 + R_{03}^2 - (2X_2X_R + 2Y_2Y_R) = R_2^2 \quad \text{(9)}
\]
\[ R_R^2 + R_{O3}^2 - (2X_3X_R + 2Y_3Y_R) = R_3^2 \] ...............(10)

Re-arrange (8)

\[ R_R^2 = R_1^2 + (2X_1X_R + 2Y_1Y_R) - R_{O1}^2 \] ...............(11)

Substitute (11) into (9) and (10)

\[ R_1^2 + (2X_1X_R + 2Y_1Y_R) - R_{O1}^2 + R_{O2}^2 + (2X_2X_R + 2Y_1Y_R) = R_2^2 \] ..............(12)

\[ R_1^2 + (2X_1X_R + 2Y_1Y_R) - R_{O1}^2 + R_{O3}^2 - (2X_3X_R + 2Y_3Y_R) = R_3^2 \] ..............(13)

From (12) find \( Y_R \) in terms of \( X_R \) and substitute into (13) to find \( X_R \)

\[
X_R = \frac{(R_2^2 - R_1^2 + R_{O1}^2 - R_{O2}^2)(Y_3 - Y_1) + (R_3^2 - R_1^2 + R_{O1}^2 - R_{O3}^2)(Y_1 - Y_2)}{2(X_2 - X_1)(Y_1 - Y_3) - 2(X_3 - X_1)(Y_1 - Y_2)}
\] ....(14)

Similarly find \( Y_R \)

\[
Y_R = \frac{(R_2^2 - R_1^2 + R_{O1}^2 - R_{O2}^2)(X_3 - X_1) + (R_3^2 - R_1^2 + R_{O1}^2 - R_{O3}^2)(X_1 - X_2)}{2(Y_2 - Y_1)(X_1 - X_3) - 2(Y_3 - Y_1)(X_1 - X_2)}
\] ....(15)
2.3.3.1) Accuracy of the position calculation process

Assuming that the correct objects have been matched and that the values of $X$ and $Y$ related to the objects were accurate only the values of $R_1, R_2,$ and $R_3$, which were obtained from the rangefinder, contained errors. This was why the calculations outlined above were used.

The ranges $R_{01}, R_{02}$ and $R_{03}$ to the centres of the features were interpolated from the ranges to the two edges. If the identification of edges was inaccurate because of non-ideal shapes then the measured ranges were likely to be inaccurate. The measurements consisted of two parts, a measured distance to the mid-point of the object side plus half the detected width, as all objects were assumed to be symmetrical (see figure 2.13). If this was untrue then an error was introduced into the equations. For larger rectangular objects the error in the predicted centre could be large and was possibly the limiting factor of the system.

Consider the object in figure 2.14. Robot 1 detected the narrow end and assumed that the object was smaller than it was and the projected centre did not protrude far enough into the object. Robot 2 detected the same object which appeared wider and gave a predicted centre outside the object. Therefore the reliability of the "range to centre" data decreased as the object deviated from the ideal. The highest reliability was placed on those objects which had a small width and were at the maximum range, as then the predicted distance from the surface to the centre was the smallest percentage of the "range to centre" value. Three such fixes produced the most accurate calculation.

If more than three separate obstacles were detected further re-calculations of the robot's position were performed using all the obstacles, which produced a series of results which were suitable for further processing.
Figure 2.13 A symmetrical object

Figure 2.14 An asymmetrical object
The estimation of position change of the mobile robot from a known starting point

In a system where the accurate position of the mobile robot was only determined at discrete intervals, there was a requirement to know the position in the interval between fixes. Assuming that an accurate position fix had been made some time in the not too distant past only the relative movement since then was required in order to calculate the present position.

There were several methods of interpolating the position of the mobile robot between position fixes. One method assumed that any movements made by the simulated mobile robot were accurately mirrored by the prototype mobile robot (see section 2.3.4.1). An alternative method internally measured the movements of the mobile robot used to update the position parameters (see section 2.3.4.2). A third method measured the change in some absolute quantity external to the mobile robot, such as the Earth's magnetic or gravitational field, which was proportional to the movements made, and then updated the position parameters (see section 2.3.4.3). A final method which was an adaptation of the previous one, measured the relative change in some external quantity which may be valid only over a "small" area, e.g. the spacing of overhead light fittings in a warehouse (see section 2.3.4.4).

The calculation of change in position using assumed movement parameters

The simulation of the mobile robot was constructed so that the position was always known and a movement of ten metres resulted in a change of position of ten metres. No sensors were required as this was an inherent function of the simulation. When a relative movement of the prototype mobile robot was required the use of movement sensors was avoided by ensuring the same property.
This required calibration of the movements of the prototype mobile robot with respect to the movements of the simulated mobile robot. The calibration process was simplified if the number of possible movements was restricted. The minimum number of movements was two; translation in a straight line and stationary rotation.

The advantage of this technique was that no sensors or additional equipment were required for correct operation. The disadvantage was that should the calibration fail or the prototype mobile robot become displaced from the simulated position (i.e. physically moved by an external force), this was not detected or rectified until the next absolute position fix. This situation resulted in the simulated and prototype mobile robots becoming disjointed and the prototype may then appear to operate in an illogical manner.

Also there was the accumulation of position errors caused by the inaccuracy of the calibration which was used to trigger the absolute position fix technique (section 2.2) whenever it exceeded a specified threshold. This ensured that the position fixing process which was resource expensive was only used when required.

2.3.4.2) The calculation of the position change using internally measured movement parameters

The technique outlined in section 2.3.4.1 was dependent on the prototype mobile robot executing accurately the specified movement. This dependency was reduced by internally measuring the change of the position of the prototype mobile robot [Myer 1971, Kraft 1978, Tsumura et al 1981].

Such a measurement system counted wheel revolutions with optical encoders and by continuously measuring the distance moved by each wheel it was possible to approximate the
change in position and orientation. [Tsumura et al 1978, Tsumura et al 1981]. Calibration was required but only for the wheel encoders, not for the whole robot. Cumulative errors occurred due to the inaccuracy in calibration and also to the algorithms used to convert the sensor data into relative position changes.

The advantages were that any incomplete movements were detected, the mobile robot movements were not restricted and any movements caused by external forces were detected provided that the wheels remained in slip free contact with the ground. The disadvantage was that additional equipment, i.e optical shaft encoders, was required.

2.3.4.3) The calculation of change of position by measuring an absolute external quantity

Methods of determining the distance travelled by measuring an absolute external quantity were not yet available. However there were devices capable of providing an absolute measurement of orientation, of which there were two main types in use; the inertial compass [Kotzin and van den Heuvel 1978] and the magnetic compass [Lezniak et al 1977]. The magnetic compass measured the orientation of the vehicle with respect to the Earth's magnetic field, and the inertial compass measured the orientation of the vehicle with respect to a rapidly spinning or vibrating mass whose initial orientation was known.

The magnetic compass had several drawbacks caused by the weakness of the Earth's magnetic field which required an undamped system and therefore had long response time and overshoot. Also the susceptibility to local distortions in the field by man-made and natural phenomena reduced the accuracy and reliability.
The inertial compass did not suffer from the drawbacks of the magnetic compass but did depend on the accuracy and reliability of the mechanical components. However, provided that a suitable device was constructed with a high reliability and low cost the inertial compass was a useful method of measuring the change in orientation of the prototype mobile robot. The output from the inertial compass was a direction relative to a known fixed initial value thereby enabling the present orientation of the mobile robot to be calculated.

The advantage of using a compass was that the accuracy of the position measurements was increased and therefore the time between absolute position fixes was also increased. The disadvantages were that extra equipment was required, it did not provide a measurement of the distance moved and calibration was difficult with the added problem that the inaccuracy of the output of the inertial compass increased with time whereas with most other types of sensors the inaccuracy increased with the distance moved.

2.3.4.4) Correction of calculated position using relative trends of the environment

Although the various techniques outlined above in sections 2.3.2.1, 2.3.2.2 and 2.3.2.3 allowed the position of a moving vehicle to be calculated, they were subject to cumulative errors between position fixes. It was possible to make use of the ordered layout of the environment to apply corrections, [Lezniak et al 1977] to the cumulative errors thereby increasing the time between absolute position fixes. The process operated by identifying a temporary trend in the environment and was usually only possible in man-made environments.

The trends had to cover a significant proportion of the operating environment and be easily identifiable. For
example if road directions were used [Lezniak et al 1977] then the trend was apparent in an area where the road axes were parallel to within a few degrees. In a factory environment a suitable trend might be the spacing of light fittings on the roof of a warehouse provided they covered the whole area and were accurately aligned and spaced.

The disadvantage of this system was that it did not operate in environments where the trends were not detectable and was therefore only operated as a secondary or backup system to one of those techniques outlined in sections 2.3.4.1 or 2.3.4.2.

2.3.4.5) Conclusion

After a consideration of the suitable techniques for measuring relative position changes of the mobile robot the conclusion was reached that the "assumed movement parameter" (see section 2.3.2.1) method was the most satisfactory for implementation on the prototype mobile robot. This was because the accuracy was satisfactory provided that the calibration processes were accurate (see section 2.3.4.6) and additional equipment was not required. Also the path finding algorithm only required the minimum two movements, rotation and translation, which minimised the calibration process without restricting the operation of the mobile robot. Also the prototype mobile robot was only used in controlled environments where external forces were prevented from affecting the mobile robot.

It was noted that at some time in the future the mobile robot would require a re-examination of position calculation techniques to maintain the accuracy when the environment was less constrained.
2.3.4.6) The proposed "assumed movement parameter" calibration process

The proposed calibration of the "assumed movement parameter" position updating process was simple. The translation was calibrated for two effects, linearity and accuracy. These were determined by mounting a pen on the prototype mobile robot and executing a translation. The trace produced was checked for linearity by a straight edge and the accuracy by measurement of the trace length. Adjustments were made and the process repeated until the required accuracy and linearity were obtained.

Rotational accuracy was obtained by executing a translation, followed by a rotation of the required angle, followed by another translation. The trace produced by this set of movements enabled the rotation angle to be measured and any necessary adjustments to be made. The process was repeated until the required accuracy was obtained.

Finally the calibration was tested by moving along a rectangular path turning anti-clockwise at the corners. The start and finish of the trace should coincide and the rectangle should be square. This process was then repeated but with clockwise turns at the corners. If necessary the calibration process was repeated until the required accuracy was achieved.

2.4) ENVIRONMENT SENSORS

If the environment was predictable to a high degree of accuracy, and the actions of the mobile robot were predictable to that same high degree of accuracy, there was no need for sensors of any description. However for a mobile robot to perform work in a totally unknown environment there was a need to sense the environment [Briot et al 1981, Masuda and Hasegawa 1981]. The robot sensed the environment to close the pathfinding and following feedback loops and it was these types of sensors which were of particular interest.
For a mobile robot, the environment property that was of most interest was the distance to the nearest object in a specified direction. This information was then used in the environment modelling (section 2.1), pathfinding (section 2.2) and position finding (section 2.3) processes. The sensors divided into two main categories, those which directly measured the range and those which calculated the range indirectly, e.g. stereo cameras, triangulation.

The following sections describe the principles of the two different types of sensors and consider the advantages and disadvantages of each with respect to the prototype mobile robot. A conclusion was reached (see section 2.4.4) and the environment sensors to be implemented were proposed (see section 2.4.5).

2.4.1) Rangefinders

A self-contained direct rangefinder operated by transmitting a signal and measuring the time taken for the reflected signal to be detected. Assuming a constant propagation velocity of the signal the distance to the object could be calculated. The transmission medium determined the coupling to the environment and effectively an unbalanced system was required where the signal was reflected rather than absorbed by the target.

Two main direct range measurement techniques were applicable to all types of rangefinders. The first measured the phase shift between the transmitted and reflected signals by modulating the transmitting medium. The finite time required for the signal to travel to the target and back again introduced a phase shift into the received signal proportional to the distance. This was detected and the range calculated.
The alternative technique used a signal consisting of a single pulse or pulse burst and measured the time of flight of the signal directly. To maintain a suitable signal to noise ratio, high peak power and low average power pulses were used. Assuming the pulse travelled at a constant velocity the measured flight time was proportional to the distance. If a threshold value was exceeded in the receiver the reflected signal was considered to have been detected. The threshold was selected to prevent "electrical noise" from causing false range values.

If detecting the presence of a signal above a certain threshold level was not sufficiently accurate due to variations in the rise times the measurement of pulse centres was used.

The measurement technique used was determined by the transmitting medium (optical, sonic, or electromagnetic) and the range required. Ranges of hundreds to thousands of kilometres required electromagnetic techniques (see section 2.4.1.3) and pulse or frequency shift measurements could be used. Optical devices (see section 2.4.1.1) had maximum ranges from a few metres up to a few kilometres and ultrasonic rangefinders (see section 2.4.1.2) generally had a maximum range of less than ten metres.

2.4.1.1 The laser rangefinder

Both pulse and continuous wave techniques were used for laser rangefinders and with the introduction of laser diodes small devices have been constructed [Lewis and Johnston 1977]. However, because of the very high velocity of light very fast pulses were required. For example, taking the speed of light to be approximately 30 cm/ns then for a rangefinder with a nominal resolution of 0.5 cm in a 10m maximum range the minimum detectable time difference between two signals must be 33 ps. Alternatively, amplitude
modulation of a continuous wave laser allowed the measurement of time to be changed to a measurement of phase, at much lower frequencies (see section 2.4.1).

To prevent damage to the environment a low average transmitted power must be used (BS 4883) with the result that statistical averages of several measurements were necessary to produce a reliable range value [Kim and Shen 1981]. Rangefinders using these techniques were complex making them unsuitable for the prototype mobile robot.

2.4.1.2) The ultrasonic rangefinder

An alternative to the laser rangefinder was the ultrasonic rangefinder, which used high frequency sound as the transmission medium. Sound travels at approximately 330m/s and a direct time of flight measurement with a resolution of 0.5 cm gave a detectable time resolution of 1.5 μs. The phase shift technique appeared to have no discernible advantage and was more complex.

Most ultrasonic transmitters were low power devices and this, combined with the attenuating effect of air, reduced the effective range. Also because of the relatively long wavelength of the pulses, the reflected signal was specular rather than dispersive so that only targets with a surface orthogonal to the transmitted signal reflected sufficient signal back to the detector. Also insufficient acoustic impedance mismatch between the acoustic energy and the targets resulted in some targets (e.g. foam) absorbing the energy and insufficient was reflected, again limiting the use and reliability of ultrasonics.

2.4.1.3) The electromagnetic rangefinder

The electromagnetic rangefinder (i.e. RADAR) was suitable for long distance measurement of high mass targets and
although theoretically possible for use in short range rangefinders, the technical problems of high speed detection electronics, electrical interference, and the lack of reflected signal from low mass targets, made this device unsuitable for the prototype mobile robot.

2.4.1.4) The scanning rangefinder

A single range measurement was of little use in this situation, as the information concerning the environment was small. However, by obtaining adjacent range measurements considerably more information was obtained, particularly by differentiation and other signal processing and recognition techniques (see section 2.3) [Nakamura 1981]. Two techniques were available for obtaining adjacent readings; one was to mechanically scan the transmitted signal, and the other was to use a spherical signal wavefront with an electrically scanned array of receivers [Tachi et al 1982].

For optical rangefinders mechanical scanning was usually achieved by the use of mirrors which were mechanically oscillated or revolved, so that the transmitted beam was scanned over a limited aperture [Nakamura 1981]. In this way a three dimensional range map of the section of environment within the field of view may be constructed [Longendorfer 1976, Kim and Shen 1981]. The mechanical scanning devices were difficult to manufacture with the required scanning speeds and were considered unsuitable for the prototype mobile robot. Electronically scanned optical beams were not yet available.

With ultrasonic rangefinders it was normal, because of the relatively long time of flight of the pulses, to rotate the transmitter and receiver pair [Seals 1985] when mechanically scanning the beam. The drawback was the time taken to make a single scan. For example, assuming a maximum range of 10m and an angle between measurements of 1.8°, a 180° scan would take 6.6 seconds.
A three dimensional scan required several scans at varying orientations and took considerably longer. For the mobile robot the scan time was too long and a faster scanning technique was required, such as electronically scanning the acoustic beam [Tachi et al 1982].

2.4.1.5) The tactile environment sensor

It was possible to gather information concerning the environment with tactile sensors, the principle being that of the rangefinder. By fixing a switch, which was closed by slight pressure, onto a rotating extendible arm, a two dimensional scan was possible [Marce et al 1980, Marce et al 1981]. The range was limited by the maximum length of the arm and considerable time was required for a complete scan. The readings obtained were similar to those from a short range rangefinder.

2.4.2) The stereo camera

An alternative to direct range measurements, and one in which whole segments of the environment were analysed, was to use two television cameras which viewed the same scene but were slightly displaced [Nevatia 1982].

Stereo depth measurement required two operations to be performed; the matching of the same point in the two images, called the correspondance problem, and the determination of the range of the point from the two views by triangulation.

Correspondance between the two views was established by matching specific features [Moravec 1979], such as corners, which had distinct orientation and position parameters. Obtaining the correspondance and eliminating ambiguities was costly in terms of time and computing resources. The computer resources required and the complexity of the problem made this technique unsuitable for implementation on the prototype mobile robot.
2.4.3) Conclusion

None of the sensors discussed in the previous sections were particularly suitable for the mobile robot as they all required a relatively long time to obtain the environment data. The most suitable device would be a high speed electronically scanned rangefinder which was not available.

However, to produce a working prototype mobile robot it was considered that a mechanically scanned ultrasonic rangefinder was the best alternative. It was simple to operate, required few resources, and by a careful optimisation of the prototype mobile robot it was possible to reduce the dependency of other functions on the time taken to scan the environment.

In addition "bump" sensors were required to detect the collision of the perimeter of the prototype mobile robot with any undetected object to prevent damage to the environment or the robot.

2.4.4) The proposed environment sensor: a mechanically scanned ultrasonic rangefinder

The long scanning time of the rangefinder was to be overcome by maintaining the prototype mobile robot stationary while the scanning and processing of data was performed. The rangefinder was to be constructed with a resolution of 0.5 cm with a maximum range of 10 metres. The scanned sector angle and intersample angle was to be alterable to allow speed and accuracy optimisation in the different situations which would occur.

A second type of sensor was proposed (although this was not implemented) to detect the collision of the perimeter of the prototype mobile robot with the environment in order to determine the point and direction of contact.
2.5) SUMMARY

This chapter has considered previous solutions to four of the processes identified in chapter one as being of important in the construction of a prototype mobile robot; modelling the environment, path selection and following, position location, and environment sensors.

Suitable solutions were then proposed for implementation on a prototype mobile robot which consisted of some form of scanning rangefinder, a polygonal model of the environment, the Means-end path selection and following and the calculation of the position of the mobile robot from three recognised environment features.
CHAPTER THREE

THE INITIAL MODELLING AND SIMULATION OF THE MOBILE ROBOT

When attempting to solve a complex problem such as a mobile robot, it was useful to model and simulate the processes involved. Solutions were then tested before physical implementation to discover any faults or flaws. The process was not guaranteed to find all the problems because the model in the simulation was never a completely accurate representation as some simplifications and compromises had to be made. Care was taken to ensure that the resources devoted to the simulation did not outweigh the expected gains.

The aim of the simulation was to mimic the real device in known situations and then to predict the actions of the process in circumstances not yet encountered and to discover what effect external conditions had and whether they were beneficial.

The initial simulation of the mobile robot chassis was simple so that the physical characteristics of the mobile robot had little effect when moving. When simulating the actions of a mobile robot the "action stimuli" were the objects in the environment, and the "reactions" to the action stimuli were the decisions controlling the mobile robot. The proposed decision-making process was described in detail in section 2.2. The following sections describe how the environment was simulated, (see section 3.1), how the decision-making process was implemented and the results obtained (see sections 3.2 and 3.3). Finally recommendations were made for an improved simulation to be implemented on the prototype mobile robot.
Three functions were required to be simulated; the environment and obstacles (see section 3.1.1), the position location of the mobile robot (see section 3.1.2) and the environment sensors (see section 3.1.3). The functions used were implemented in Pascal [Jensen and Wirth 1978] (see Appendix A for a program listing).

3.1.1) The implementation of the grid environment

A detailed analysis of possible environment representations was made in chapter 2, and the grid format was chosen for its simplicity and ease of implementation. A direct mapping between the environment and model was required, which was simple and easy to use. A two dimensional array system allowed plan representations and explicitly defined each square to contain either free space, an obstacle, the robot, or any other designation valid at the time.

A coarse quantisation of the environment was used, with the minimum grid size being taken as representing the physical size of the robot due to the memory constraints imposed. A nominal array of 120 x 120 locations was used giving the robot a reasonable amount of movement and allowing a number of obstacles. It also permitted the array to be printed out directly on any printer with a linewidth of greater than 120. A portion of the array of 23 x 80 could be used for test purposes, as this could be completely displayed on the VDUs available, i.e. ADM 3+ [Lear Siegler 1979].

Several copies of the environment array were maintained with varying degrees of accuracy where the accuracy here means the degree of similarity between the array undergoing processing and the fixed array representing the physical environment. Two were of major significance CODE1ARRAY and ROBOT1ARRAY. CODE1ARRAY represented the physical environment
where a grid square containing a "1" was modelling an obstacle, and an empty grid square represented an unobstructed area. No other characters were allowed in CODE1ARRAY except for the simulated mobile robot representation which was the character "X".

For the remainder of this chapter the environment and mobile robot are referred to as the model environment and the model mobile robot as no real situations were investigated. The model mobile robot also maintained a simulated version of the environment and mobile robot, and these are referred to as the simulated environment and the simulated mobile robot. If the model or simulation prefix are not used this means both situations are applicable.

The second array ROBOT1ARRAY contained the environment model that was used by the simulated mobile robot when planning paths. If ROBOT1ARRAY was completely accurate there was a one-to-one correspondence between CODE1ARRAY and ROBOT1ARRAY. To observe the effect of discrepancies in the simulated and model environments these two arrays were not maintained the same. For the majority of experiments the initial state of ROBOT1ARRAY was that of a completely obstacle free area.

As a visual aid to monitoring the progress of the simulated mobile robot in sensing the environment an unscanned free space in ROBOT1ARRAY was denoted by a full stop, ",", and an unscanned obstacle by a hash, "#". As the simulated mobile robot moved the contents of the grid squares were "sensed" by the rangefinder and bump sensors, the data being transferred from CODE1ARRAY into ROBOT1ARRAY. An example of CODE1ARRAY is shown in figure 3.1 and of ROBOT1ARRAY in figure 3.2. The environment shown in figure 3.3 is an example of ROBOT1ARRAY after the simulated mobile robot has moved. For three grid spaces surrounding the path of the simulated mobile robot full stops have been converted into
Figure 3.1 An example of CODE1ARRAY

Figure 3.2 An example of ROBOT1ARRAY
Figure 3.3 An example of ROBOT1ARRAY after the mobile robot has moved

```plaintext
117  0  0  PROCEDURE  INITIALISE;
118  0  0
119  0  1  BEGIN
120  0  1  MOVE(DESTINATION);
121  1  1  STACKPOINTER := 1;
122  15  1  DIRECTION[STACKPOINTER] := NONE;
123  38  1  FOR  J := 1 TO M DO
124  54  1  FOR  I := 1 TO L DO
125  72  2  BEGIN
126  74  2  CODE1ARRAY[I,J] := ' ';
127  117  2  ROBOT1ARRAY[I,J] := ' ';
128  160  2  OBJARRAY[I,J] := '.';
129  203  2  END;
130  221  1  XCOORDIN := 8;
131  225  1  YCOORDIN := 8;
132  229  1  FOR  J := YCOORDIN-3 TO YCOORDIN +3 DO
133  249  2  BEGIN
134  251  2  FOR  I := XCOORDIN-3 TO XCOORDIN +3 DO
135  271  2  ROBOT1ARRAY[I,J] := CODE1ARRAY[I,J];
136  372  2  END;
137  381  1  ROBOT1ARRAY[XCOORDIN,YCOORDIN] := 'X';
138  424  1  END; (* INITIALISE *)
```

Figure 3.4 The procedure INITIALISE
ject edges from hashes into ones. (Due to the simulated rangefinder the area behind obstacle perimeters appeared to be scanned but this did not significantly affect the operation of the simulation).

The paths through the grid models of the environment were not explicitly visible in CODE1ARRAY and ROBOT1ARRAY and in order to achieve this a third environment array, RESMATRIX, was used. RESMATRIX was used as the environment representation which was to be printed, to provide a "window" into the internal state of the simulation. Initially, ROBOT1ARRAY (or OBJSAARRAY which was an alternative copy of CODE1ARRAY) was copied into RESMATRIX and then as movements were executed, both ROBOT1ARRAY and RESMATRIX were updated with RESMATRIX containing additional symbols denoting the past path of the simulated mobile robot.

To provide flexibility the path finding procedures used general environment names, FROMARRAY for the environment the sensors operate in (i.e. the model of the physical environment) and FINALARRAY for the array where the sensor data was stored. Therefore, by equating FROMARRAY and FINALARRAY to ROBOT1ARRAY simulated paths through the environment could be planned and, by then equating FROMARRAY to CODE1ARRAY, the simulated mobile robot was able to follow the planned path through the model of the physical environment. If CODE1ARRAY and ROBOT1ARRAY differed there was the possibility of encountering obstacles which were not yet present in ROBOT1ARRAY. ROBOT1ARRAY was updated from CODE1ARRAY using the data from the simulated sensors when the model mobile robot was moving. TOARRAY was used as a temporary store for sensor data during the processing before FINALARRAY was updated.

The following explanations describe the PASCAL procedures involved with manipulating the environment. Where lines
previously named procedure can be found in Appendix A between the line numbers specified.

COPY (lines 100-107) copied the specified array into another specified array. CLEAR (lines 109-115) filled the specified array with space characters. INITIALISE (figure 3.4) initialised the environment by filling CODE1ARRAY and ROBOT1ARRAY with spaces and OBJARRAY with dots. PRINTARRAY (lines 326-345) printed the required portion of the specified array onto the VDU using X-Y addressing of the character positions for dynamic viewing of the simulation and used the procedure GOTOYX. GOTOYX (lines 309-324) moved the VDU cursor to the specified position using the X-Y addressing capability of the VDU (which only worked correctly for ADM 3+ VDU) and used the procedure PUTASC. PUTASC (lines 300-307) output the specified character to the VDU avoiding the existing PASCAL output procedures. CONSULT (lines 347-367) decided which array, CODE1ARRAY, ROBOT1ARRAY or OBJARRAY to output to the VDU. GETDESTINATIONS (lines 1207-1234) specified the points the simulated mobile robot was to visit in the environment, saving the values in a file called ENVIRON. GETOBJECTS (lines 1236-1269) added the perimeter of the model environment then entered all the required objects.

The environment was bounded by a perimeter "wall" on all four sides of the array to prevent the rangefinder from failing at the perimeter and the simulated mobile robot from leaving the specified environment area. Objects were specified by the top left hand vertex coordinates (as viewed on the VDU), the length in the X axis direction, and the width in the Y axis direction. (Note: the positive Y direction was from the top of the screen downwards to match the X-Y addressing technique of the VDU.) Only rectangular objects were allowed with other shapes being "built" up from smaller rectangles.
put the specified object into CODEIARRAY as "l"'s and OBJARRAY as "H"s. ADDPLACES (lines 224-232) put the specified point into the destination list using the procedure ENTERPOINT and denoted that it was a destination to be visited by entering "+" into OBJARRAY. GETENVIRON (lines 1165-1205) added all the objects into CODEIARRAY and OBJARRAY, and the destinations into the destination list, taking the information from the file ENVIRON. CONDITIONS (lines 1460-1501) set up the entire program and determined whether new or existing environment models were to be used.

3.1.2) The implementation of the simulated mobile robot and position location functions

As the aim of the simulation was to experiment with the pathfinding algorithms other functions were simplified as much as possible. The simulated mobile robot was considered to occupy a single grid square and could only move into one of the eight adjacent squares (see figure 3.5). The simulated mobile robot was prevented by the bump detector from moving into a grid square already occupied by an obstacle. No other physical characteristics of the mobile robot were simulated and acceleration, deceleration, motoring, cornering, radius or orientation values were neglected or not used. All movements were considered to be instantaneous from one grid square to the next and the simulation did not operate in real-time.

There was no error in the position values of the simulated mobile robot as the location of the mobile robot was always defined by two integer values, XCOORDIN and YCOORDIN.
Figure 3.5 The possible moves of a mobile robot in a grid environment

Figure 3.6 The area scanned by the rangefinder in the grid environment
3.1.3) The implementation of the scanning rangefinder and bump detector

As the environment was represented by a regular grid structure a 360° scanning rangefinder was not simple to implement if only those edges of objects nearest to the simulated mobile robot were to be detected. The rows of squares along the eight movement axes (see figure 3.6) were easy to scan. First the nearest to the robot was checked and if empty the next furthest away was checked, and so on. If one of the grid squares scanned in this way contained an obstacle it had to be the obstacle perimeter. Successive grid squares further away then could not be scanned (see figure 3.7).

This process was performed by the procedure SEARCHAXIS (lines 539-591) which simulated the action of the rangefinder for three grid squares' distance along the specified movement axis. The data was temporarily copied from FROMARRAY (which was either CODE1ARRAY or ROBOT1ARRAY) into TOARRAY unless prevented by a detected edge.

As the range increased it became apparent that grid squares had not been scanned between adjacent axes of movement (see figure 3.6). SEARCHAREA scanned the single grid square between two axes, nearest to the simulated mobile robot. If it did not contain an obstacle LOOKATAREA (lines 418-537) was used to copy the remaining unscanned grid squares into TOARRAY. Any obstacle perimeters encountered in the outer layer of grid squares had no effect as the rangefinder was limited to a maximum range of three grid squares. Without additional procedures the maximum range of the rangefinder could not be extended. The 360° scan was controlled by the procedure EDGEDETECT (see figure 3.8) which used SEARCHAXIS and SEARCHAREA to simulate a scan in all the available directions.

The bump sensor was simulated by the procedure INQUIRE (see figure 3.9) and was only active in the eight grid squares immediately surrounding the position of the simulated mobile
Figure 3.7 The effect of objects on the rangefinder scan

Figure 3.8 The procedure EDGEDETECT
PROCEDURE INQUIRE (VAR DIRECTION2:MOVEMENT);
VAR
TEMPX,TEMPY:INTEGER;
BEGIN
TEST1;
OBSTRUCT := FALSE;
TEMPX := XCOORDIN;
TEMPY := YCOORDIN;
CASE DIRECTION2 OF
NONE :;
NORTH : TEMPY := TEMPY - 1;
SOUTH : TEMPY := TEMPY + 1;
WEST : TEMPX := TEMPX - 1;
EAST : TEMPX := TEMPX + 1;
NEAST : BEGIN
TEMPX := TEMPX + 1;
TEMPY := TEMPY - 1;
END;
NWEST : BEGIN
TEMPX := TEMPX - 1;
TEMPY := TEMPY - 1;
END;
SEAST : BEGIN
TEMPX := TEMPX - 1;
TEMPY := TEMPY + 1;
END;
SWEST : BEGIN
TEMPX := TEMPX - 1;
TEMPY := TEMPY + 1;
END;
THEN XFIRST(PRIMARY);
ELSE YFIRST(PRIMARY);
END;
END;
TEMPX := TEMPX - 1;
TEMPY := TEMPY + 1;
END;
CASE TEMPCHAR OF
'0' : OBSTRUCT := FALSE;
'1' : OBSTRUCT := TRUE ;
'X' :;
'+' :
'X':
END;
END;
END; (*CASE*)
END; (*INQUIRE*)

PROCEDURE WHICH;
BEGIN
COUNTINT := 0;
COUNTAXIS := 0;
COUNTOTAL := 0;
AXISEND := FALSE;
ARRIVE := FALSE;
FIRSTIME := TRUE;
IF STACKPOINTER < 2 THEN
BEGIN
XOBJECT:=DESTINATIONA.POSITION.XPOSITION;
YOBJECT:=DESTINATIONA.POSITION.YPOSITION;
IF ABS(XCOORDIN-XOBJECT)=ABS(YCOORDIN-YOBJECT)
THEN DIRECTION[STACKPOINTER]:=NONE;
ELSE XFIRST(PRIMARY);
END;
END; (*WHICH*)

Figure 3.9 The procedure INQUIRE

Figure 3.10 The procedure WHICH
robot, and used the information obtained by the rangefinder in TOARRAY. If the grid square the simulated mobile robot was about to move into was empty, the move was completed by updating the values of XCOORDIN and YCOORDIN, and copying the grid squares from TOARRAY into FINALARRAY. If there was an obstacle the move was aborted, the information in TOARRAY discarded and the obstacle avoidance procedures executed. TOARRAY was necessary to prevent data in FINALARRAY being overwritten as otherwise data retrieval was not possible if a movement was aborted.

3.2) IMPLEMENTATION OF THE MEANS-END PATH FINDING ALGORITHM

The implementation using the two dimensional grid model of the environment of the Means-end path finding algorithm (section 2.2.2.7) consisted of three basic functions; avoiding an obstacle which was in the path of the simulated mobile robot (see section 3.2.1), finding an initial path through an environment (see section 3.2.2) using the obstacle avoidance function, and improving and following the initial path (see section 3.2.3).

Once these functions had been implemented the effect of various environment states, known, unknown and partially known, was investigated (see section 3.2.4) and some conclusions drawn as to the suitability of this algorithm for the prototype mobile robot.

3.2.1) Avoiding an obstacle

The first action to be taken was to decide in what direction the mobile robot should move to reach the destination, as there were an infinite number of possible paths between two points. From the point of view of the mobile robot the shortest was preferred and this was the straight line joining the two points. In a grid environment model a straight line path was composed of a combination of
movements in two directions, one of which was an axial movement (denoted by North, South, East or West) and the other a diagonal movement (denoted NorthEast, SouthEast, NorthWest or SouthWest).

The selection of directions was performed in the following manner. First the difference between the two X and Y coordinates was calculated.

\[ X_{\text{diff}} = |X_{\text{robot}} - X_{\text{destination}}| \]
\[ Y_{\text{diff}} = |Y_{\text{robot}} - Y_{\text{destination}}| \]

The difference between \( X_{\text{diff}} \) and \( Y_{\text{diff}} \) gave the number of grid squares along an axis, and the smaller value the number of grid squares along a diagonal axis, to produce the shortest path between the mobile robot position and the destination.

\[ N_{\text{axis}} = |X_{\text{diff}} - Y_{\text{diff}}| \]

\[ N_{\text{diagonal}} = X_{\text{diff}} \text{ if } X_{\text{diff}} \leq Y_{\text{diff}} \]
\[ \quad \text{or } Y_{\text{diff}} \text{ if } Y_{\text{diff}} < X_{\text{diff}}. \]

Tables 3.1 and 3.2 show how the combination of the signs and relative values of \( X_{\text{diff}} \) and \( Y_{\text{diff}} \) determined which particular axial and diagonal directions were required.

There was an additional direction called NONE used to indicate that no movement was required: when \( X_{\text{diff}} = Y_{\text{diff}} \) no axial movement was necessary, or when \( X_{\text{diff}} \) or \( Y_{\text{diff}} = 0 \) no diagonal movement was necessary.
Table 3.1 Diagonal Selection

<table>
<thead>
<tr>
<th>$X_{\text{diff}}$</th>
<th>$Y_{\text{diff}}$</th>
<th>Diagonal</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
<td>NE</td>
</tr>
<tr>
<td>+</td>
<td>−</td>
<td>SE</td>
</tr>
<tr>
<td>−</td>
<td>+</td>
<td>NW</td>
</tr>
<tr>
<td>−</td>
<td>−</td>
<td>SW</td>
</tr>
</tbody>
</table>

Table 3.2 Diagonal Selection

<table>
<thead>
<tr>
<th>Diagonal</th>
<th>Largest difference</th>
<th>Axis required</th>
</tr>
</thead>
<tbody>
<tr>
<td>NE</td>
<td>$X_{\text{diff}}$</td>
<td>E</td>
</tr>
<tr>
<td>NE</td>
<td>$Y_{\text{diff}}$</td>
<td>N</td>
</tr>
<tr>
<td>SE</td>
<td>$X_{\text{diff}}$</td>
<td>E</td>
</tr>
<tr>
<td>SE</td>
<td>$Y_{\text{diff}}$</td>
<td>S</td>
</tr>
<tr>
<td>NW</td>
<td>$X_{\text{diff}}$</td>
<td>W</td>
</tr>
<tr>
<td>NW</td>
<td>$Y_{\text{diff}}$</td>
<td>N</td>
</tr>
<tr>
<td>SW</td>
<td>$X_{\text{diff}}$</td>
<td>W</td>
</tr>
<tr>
<td>SW</td>
<td>$Y_{\text{diff}}$</td>
<td>S</td>
</tr>
</tbody>
</table>
The procedures which implemented these calculations were NEGXAXIS, NEGYAXIS, POSXAXIS, POSYAXIS, XFIRST and YFIRST (lines 705-757) controlled by the procedure WHICH (see figure 3.10).

The ratio of axial to diagonal movements was calculated and implemented by the procedure NEXTMOVE (see figure 3.11). The simulated mobile robot moved along the approximation of a straight line between the start and destination one grid square at a time.

The check of the next grid square to be moved into to ensure that it was obstacle free was controlled by the procedure LEVEL1 (see figure 3.12) which used NEXTMOVE to indicate the next grid square and the bump detectors to detect the obstacle. If an obstacle was detected the next level of obstacle avoidance algorithm controlled by the procedure LEVEL2 (lines 1065-1116) was initiated. The essence of the obstacle avoidance algorithm was to follow the perimeter of the obstacle by moving in those unobstructed directions which reduced the distance to be travelled between the present position and the destination. This could involve moving physically away from the destination in order to negotiate the obstacle.

Although eight directions were available for movement only the four axial movements, North, South, East and West, were used for obstacle avoidance. The axial direction, called the primary direction, being used as part of the straight line approximation was checked and if obstacle free was used as the next move, when the obstacle avoidance algorithm was first initiated. If still obstructed a secondary direction was chosen by the procedure LEVEL2 which was the direction at right angles to the primary direction which still moved towards the destination. If this direction was obstructed the next level, controlled by the procedure LEVEL3 (lines 1035-1063), was executed. This checked the third direction
PROCEDURE NEXTMOVE;

VAR VAR TEMPREAL : REAL;
FINISH : BOOLEAN;

BEGIN IF FIRSTIME THEN
BEGIN IF ABS(XCOORDIN-XOBJECT) = ABS(YCOORDIN-YOBJECT) THEN
BEGIN SMALLDIFF := ABS(XCOORDIN-XOBJECT);
LAGGEDIFF := ABS(YCOORDIN-YOBJECT);
END;
ELSE BEGIN
SMALLDIFF := ABS(YCOORDIN-YOBJECT);
LAGGEDIFF := ABS(XCOORDIN-XOBJECT);
END;
END;
IF SMALLDIFF = INTERVAL THEN
BEGIN
DEPEND := LARGEDIFF/SMALLDIFF;
INTERVAL := TRUNC(TEMPREAL);
FIRSTIME := FALSE;
END;
ELSE BEGIN
IF SMALLDIFF = 0 THEN
BEGIN
DIRECTION[STACKPOINTER] := AXIS;
COUNTOTAL := COUNTOTAL +1;
END;
ELSE BEGIN
IF (COUNTINT(INTERVAL) AND NOT AXISEND) THEN
BEGIN DIRECTION[STACKPOINTER] := AXIS;
COUNTINT := COUNTINT +1;
COUNTAXIS := COUNTAXIS +1;
COUNTOTAL := COUNTOTAL +1;
END;
ELSE BEGIN
DIRECTION[STACKPOINTER] := DIAGONAL;
COUNTINT := 0;
COUNTOTAL := COUNTOTAL +1;
END;
END;
END;
IF COUNTAXIS = NUMBAXIS THEN
AXISEND := TRUE;
END;
END;
END;
END;
END;
END;
END;
BEGIN IF COUNTINT(INTERVAL) AND (NOT AXISEND) THEN
BEGIN DIRECTION[STACKPOINTER] := AXIS;
COUNTINT := COUNTINT +1;
COUNTAXIS := COUNTAXIS +1;
COUNTOTAL := COUNTOTAL +1;
END;
ELSE BEGIN
DIRECTION[STACKPOINTER] := DIAGONAL;
COUNTINT := 0;
COUNTOTAL := COUNTOTAL +1;
END;
END;
END;
END;
BEGIN IF AXISEND THEN
BEGIN
STOPl := TRUE;
END;
END;
BEGIN IF (XCOORDIN=DESTINATIONA.POSITION.XPOSITION) AND (YCOORDIN=DESTINATIONA.POSITION.YPOSITION) THEN
ARRIVE := TRUE;
END;
(*NEXTMOVE*)

PROCEDURE LEVEL1;

VAR VAR DIFFERENCE : INTEGER;

BEGIN IF DIFFERENCE = ABS(XCOORDIN-XOBJECT) +ABS(YCOORDIN-YOBJECT) THEN
BEGIN
END;
END;
BEGIN IF DIFFERENCE < 3 THEN
BEGIN
END;
END;
BEGIN IF HAND THEN
BEGIN UPDATE;
END;
BEGIN IF NOT ARRIVE THEN
BEGIN
WRITE('NEXTMOVE',COUNTINT:3,COUNTAXIS:3,COUNTOTAL:3);
END;
END;
BEGIN IF NOTWRITE THEN
BEGIN
WRITELN('NEXTMOVE',COUNTINT:3,COUNTAXIS:3,COUNTOTAL:3);
END;
BEGIN IF ARRIVE THEN
BEGIN STOP1 := TRUE;
END;
END;
(*LEVEL1*)
which was the opposite to the secondary direction (e.g. if secondary was North then tertiary became South). Obstruction at this level caused the next level, executed by the procedure LEVEL4 (lines 1004-1033) which chose the fourth direction to be the opposite of the primary direction, and caused the mobile robot to backtrack. If still obstructed the mobile robot was almost certainly trapped in a "spiral" which required additional maze solving algorithms which were not implemented. To prevent this, the obstacle configuration was arranged to prevent "spiral" situations occurring. A simple spiral that was allowed was that formed by a room with only one door (see figure 3.13) from which the mobile robot could be extricated by moving in a fifth direction taken to be the same as the second direction (i.e. the opposite of the third) and was executed by the procedure LEVEL5 (lines 973-1002).

The process at each level was almost the same; if the grid square in the selected direction was empty the simulated mobile robot moved into it. Then checks were performed to determine whether the previous direction was still obstructed. If the previous direction was obstacle free the mobile robot moved into it and dropped to a lower level of obstacle avoidance procedure (i.e. from LEVEL3 to LEVEL2) and continued moving in that direction. If at any time the direction being moved in was obstructed, a higher level of procedure was moved to (i.e from LEVEL3 to LEVEL4) and a new direction selected. All the directions were stored on a first-in-last-out stack which had directions added or deleted, as required by the obstacle avoidance procedures. Assuming that the obstacles did not form spiral patterns this algorithm was capable of finding a path around any obstacle.

When an obstacle was finally cleared the algorithm was at LEVEL2 but did not immediately return to LEVEL1. This was because the simulated mobile robot had been forced from the
Figure 3.13 Negotiating a simple spiral obstacle

Figure 3.14 The procedure COMPARE

```plaintext
286  0  0 PROCEDURE COMPARE;
287  0  0
288  0  1 BEGIN
289  0  1 IF (XCOORDIN=OBJECT) AND (YCOORDIN=OBJECT) THEN
290  19  2 BEGIN
291  19  2  LASTPLACE := DESTINATION;
292  31  2  DESTINATION := NEXTPLACE;
293  43  2  IF NEXTPLACE <> NIL THEN
294  50  2  NEXTPLACE := NEXTPLACE^.NEXT
295  72  2  ELSE
296  72  2  NEXTPLACE := NEXTPLACE;
297  84  2  END;
298  84  1 END; (*COMPARE*)
```
selected straight line path by the obstacle, so that another straight line was necessary. This was obtained by using the same process as outlined at the beginning of this section but used the present position of the mobile robot and the destination. The present position of the simulated mobile robot was called an Intermediate Turning Point.

3.2.2) The initial path connecting the Intermediate Turning Points

An initial path through the model environment was found by using the obstacle avoidance algorithm outlined in the previous section (3.2.1) and forming a linked list of all the Intermediate Turning Points generated. Initially the linked list only contained the start and destination positions but as obstacles were negotiated the intermediate turning points were added until the destination was reached and formed the initial path.

The linked list was initialised by the procedure INITIALISE (see figure 3.4) and used the procedure STARTPLACE (lines 155-173) to enter the starting position of the mobile robot and the procedure ENTERPOINT (lines 175-222) to add the required number of destinations. Each destination had a sequence number indicating the order in which the destinations were to be visited. A particular destination was only visited after all lower numbered destinations. The procedure COMPARE (see figure 3.14) decided which was the next destination or Intermediate Turning Point in the linked list, to be visited.

The Intermediate Turning Points were generated at LEVEL2 by the procedure SHORTROUTE (lines 895-915) and used the present position of the simulated mobile robot and the sequence number of the destination, and were added to the linked list by the procedure INSERTTURN (see figure 3.15).
885 0 0 PROCEDURE INSERTTURN(TURN : TURNINFO);
886 0 0
887 0 1 BEGIN
888 0 1 NEW(NEWTURNINFO);
889 20 1 NEWTURNINFO.POSITION := TURN;
890 37 1 LASTPLACE.NEXT := NEWTURNINFO;
891 59 1 NEWTURNINFO.NEXT := DESTINATION;
892 81 1 LASTPLACE := NEWTURNINFO;
893 93 1 END;(*INSERTTURN*)

Figure 3.15 The procedure INSERTTURN

Figure 3.16 The initial path through a simple environment
Other procedures which affected the linked list were DUMP (lines 943-965) which removed all Intermediate Turning Points from the linked list if an obstacle were encountered when the model mobile robot was following the list, and LISTTURNS (lines 918-941) which printed out the linked list on the VDU or printer.

The initial planned path consisted of the linked list of destinations and Intermediate Turning Points generated by the simulated mobile robot as it avoided the obstacles along the straight line segments joining successive destinations.

3.2.3) Route following and improved paths (the effect of local diversions)

When the linked list was used by the simulated mobile robot to move in straight line segments from the start to the destination there was a high probability that further obstacles were encountered. For example, in figure 3.16 Start, T₁, T₂ and Finish defined the initial path found using the obstacle avoidance algorithm. Following this path resulted in the mobile robot coming into contact with obstacle 3 before the Intermediate Turning Point T₂ (see figure 3.17). However, if the path finding algorithm was executed a second time using the initial path linked list more Intermediate Turning Points were added to the linked list whenever obstacles were encountered (see figure 3.17). This produced a second "improved" path and if this path was now used by the path finding algorithm no further obstacles were encountered (see figure 3.18).

There was always the possibility that following an improved path would cause the mobile robot to encounter further obstacles, but by repeatedly executing each version of the path until no obstacles were encountered, a path from the required start position to the destination via the Intermediate Turning Points was termed obstacle free.
Figure 3.17 The second path through a simple environment

Figure 3.18 The final path through a simple environment
Assuming that the simulated environment was an accurate representation of the surrounding area and that the model mobile robot moved accurately in straight lines from one point to the next (which was always true in the simulation) then any path which was obstacle free in the simulated environment was obstacle free in the model environment.

A problem with the repeated execution of a path was that additions to the path increasingly represented solutions to the local obstacle avoidance problem which was a tactical one, rather than the global or strategic problem of moving from the start to the destination. The result was that repeated executions added little to the path itself but required the same resources as the initial path. In the simple environments tested (see section 3.3) it was found that the maximum number of times a path was executed before being found to be obstacle free was three. Therefore, to allow for more complex environment situations the number of path improvements allowed was seven. This limit of seven was not reached for any of the environments under examination.

One further point was that if the model mobile robot was executing the obstacle free path found from the simulated environment and an obstacle was encountered, then the simulated environment in ROBOT1ARRAY was considered to be inaccurate and the planned path of no use. The Intermediate Turning Points were therefore removed from the linked list by the procedure DUMP whenever this occurred. No method of attempting to link into the path more Intermediate Turning Points further away from the point of contact with the obstacle as proposed in an earlier section (2.2) was implemented. The model mobile robot then operated the obstacle avoidance algorithm as it was moving in the same manner as the initial path was found.

Two further procedures were required: BRANCH (see figure 3.19) which moved the simulated mobile robot in the
PROCEDURE BRANCH(DECISION : TYPEOFDECISION);
BEGIN
STORESTART(XCOORDIN,YCOORDIN);
TRIALNUMB := ONE;
REPEAT
BEGIN
STOPBLOCK1 := FALSE;
RESETSTART;
METOBJECT := FALSE;
WHILE NOT STOPBLOCK1 DO
BLOCK1(ROBOT1ARRAY,DISPLAY,ENABLE,ROBOT1ARRAY,DECISION);
END;
RESULTS:
TRIALNUMB := SUCC(TRIALNUMB);
UNTIL NOT METOBJECT OR (TRIALNUMB = SEVEN);
END; (*BRANCH*)

PROCEDURE ROUTEFOLLOW(DECISION : TYPEOFDECISION);
BEGIN
CONSULT(DISPLAY);
RESETPROG;
RESETSTART:
STOPBLOCK1 := FALSE;
DUMPED := FALSE;
TRIALNUMB := XX;
WHILE NOT STOPBLOCK1 DO
BLOCK1(CODE1ARRAY,DISPLAY,DISABLE,ROBOT1ARRAY,DECISION);
CONSULT(DISPLAY);
RESULTS;
END; (*ROUTEFOLLOW*)
environment in ROBOT1ARRAY when repeatedly executing LEVEL0 to find an obstacle free path, and ROUTEFOLLOW (see figure 3.20) which followed the path found in BRANCH, which moved the model mobile robot in the environment in CODELARRAY.

Various other procedures were used in the simulation to implement secondary functions such as printing and displaying the results, see appendix A.

3.2.4) The effect of inaccuracy in the simulated environment

The effects of varying degrees of accuracy between ROBOT1ARRAY and CODELARRAY on the operation of the path finding and route following algorithms were considered. Known (see section 3.2.4.1), unknown (see section 3.2.4.2) and partially known (see section 3.2.4.3) environments were investigated and some sample environments and paths obtained. The model environment used in all instances was the same and was a rectangle of 120 grid squares in the X axis direction, and 40 grid squares in the Y axis direction.

3.2.4.1) The path finding and following algorithms in known environments

Three paths through a known environment (see figure 3.1) were used as examples. The first path was from (8,8) to (66,38) and the initial path was shown in figure 3.21. The first obstacle was encountered after moving five grid squares and the obstacle avoidance algorithm moved South as this path continued to move towards the destination. When the end of the obstacle was encountered the path towards the destination was clear and an Intermediate Turning Point was generated, denoted by the character "I". The second obstacle was then encountered and South was again chosen as the alternative direction. This time, however, the obstacle prevented further moves South after seven grid squares. The opposite of South (i.e. North) was then attempted until the
Figure 3.21 Path 1: the initial path

Figure 3.22 Path 1: the second path
top of the obstacle was found and movements could continue West, and a second Intermediate Turning Point was generated. A further Intermediate Turning Point was generated before the obstacle was completely negotiated because only four directions were used in the obstacle avoidance algorithm but eight directions were used in the path planning. (This was due to an incomplete implementation of the simulation.) A third and final obstacle was encountered before reaching the destination and was negotiated by moving West. Again several Intermediate Turning Points were generated along the side of the obstacle due to the difference in allowable directions. (This problem did not affect the operation of the algorithms themselves but did generate unnecessary Intermediate Turning Points.) The linked list for the initial path was as shown below:

*Initial path linked list*

<table>
<thead>
<tr>
<th>START</th>
<th>DESTINATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>(8, 8)</td>
<td>(68, 36)</td>
</tr>
<tr>
<td>ITP1</td>
<td></td>
</tr>
<tr>
<td>(13, 15)</td>
<td></td>
</tr>
<tr>
<td>ITP2</td>
<td></td>
</tr>
<tr>
<td>(23, 14)</td>
<td></td>
</tr>
<tr>
<td>ITP3</td>
<td></td>
</tr>
<tr>
<td>(26, 14)</td>
<td></td>
</tr>
<tr>
<td>ITP4</td>
<td></td>
</tr>
<tr>
<td>(58, 29)</td>
<td></td>
</tr>
<tr>
<td>ITP5</td>
<td></td>
</tr>
<tr>
<td>(60, 29)</td>
<td></td>
</tr>
<tr>
<td>ITP6</td>
<td></td>
</tr>
<tr>
<td>(62, 29)</td>
<td></td>
</tr>
<tr>
<td>ITP7</td>
<td></td>
</tr>
<tr>
<td>(66, 29)</td>
<td></td>
</tr>
</tbody>
</table>

ITP - Intermediate Turning Point.

The second path through the simulated environment following the initial path encountered no further obstacles (see figure 3.22) and the path was passed to the model mobile robot to be executed. As the simulated environment was an accurate copy of the model environment the path was followed by the model mobile robot without any problems (see figure 3.23).
Figure 3.23 Path 1: the final path

Figure 3.24 Path 2: the initial path
A second path from (68,36) to (96,23) was planned (see figure 3.24 and no obstacles were encountered so the initial path was followed directly by the model mobile robot (see figure 3.25).

A third path from (96,23) to (10,10) was planned (see figure 3.26) and three objects were encountered and three Intermediate Turning Points generated and added to the linked list. The initial path was then followed using the improved linked list (see figure 3.27) and an additional Intermediate Turning Point generated because of the last obstacle before the destination. This necessitated another path following exercise (see figure 3.28) before the planned path was considered obstacle free. The model mobile robot followed the planned path with no problems (see figure 3.29). The final linked list was as shown below:

**Final linked list of the third example**

<table>
<thead>
<tr>
<th>START</th>
<th>(96,23)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITP1</td>
<td>(90,18)</td>
</tr>
<tr>
<td>ITP2</td>
<td>(70,13)</td>
</tr>
<tr>
<td>ITP3</td>
<td>(17, 4)</td>
</tr>
<tr>
<td>ITP3₁</td>
<td>(16, 4)</td>
</tr>
<tr>
<td>DESTINATION</td>
<td>(10,10)</td>
</tr>
</tbody>
</table>

ITP3₁ - The first Intermediate Turning Point added in a subsequent path finding exercise after ITP3.

**3.2.4.2) The path finding and following algorithms in unknown environments**

In a completely unknown environment in which ROBOTARRAY contained no obstacles, the initial path (see figure 3.30) between the START at (8,8) and the DESTINATION at (66,58) were straight line approximations. No objects were
Figure 3.25 Path 2: the final path

Figure 3.26 Path 3: the initial path
Figure 3.27 Path 3: the second path

Figure 3.28 Path 3: the third path
Figure 3.29 Path 3: the final path

Figure 3.30 Path 4: the planned path
encountered and the path was passed to the model mobile robot to be executed. When the model mobile robot attempted to follow the initial path the first obstacle was encountered and the Intermediate Turning Points dumped (as indicated by the "D" in figure 3.31). The model mobile robot found a path using the obstacle avoidance algorithm. The path was similar to that found when the environment was known (see section 3.2.4.1) as expected.

A second path through the unknown environment from (66,38) to (96,23) encountered no obstacles in the simulated and model environments (see figures 3.32 and 3.33) and was followed correctly.

The situation where the simulated environment contained objects which were not in the model environment was not considered in this thesis.

3.2.4.3) The path finding and following algorithms in partially known environments

A partially known environment was produced by the movement of the model mobile robot through an unknown environment as the rangefinder transferred the environment data from the model to the simulated environments. Such an environment was produced by the two paths in the previous section (see section 3.2.4.2).

A path from (96,23) to (10,10) was planned (see figure 3.34). The first two objects were not known in the simulated environment so they were ignored, however the second two had been partially scanned and required the operation of the obstacle avoidance algorithm. The final obstacle produced a longer deviation than necessary indicating that alternative techniques might be required if the shortest path was always to be found. Two Intermediate Turning Points were generated and added to the linked list to form the initial path. The
Figure 3.31 Path 4: the path taken

Figure 3.32 Path 5: the planned path
Figure 3.33 Path 5: the path taken

Figure 3.34 Path 6: the planned path
initial path was followed by the simulated mobile robot and was obstacle free (see figure 3.35) but an interesting observation was made. The first Intermediate Turning Point visited caused a deviation from what would have been a better path, moving directly to the second Intermediate Turning Point and then to the destination, which indicated that the method of path planning used would not automatically find the shortest path even when obstacles were correctly negotiated.

When the model mobile robot followed the path given by the linked list, an obstacle was immediately encountered and all the Intermediate Turning Points were dumped. A path was then found using the obstacle avoidance algorithm and data from the rangefinder and bump detectors (see figure 3.36).

Once the second previously unknown obstacle had been negotiated the path could have been improved by moving towards the second Intermediate Turning Point generated by the initial path, rather than the destination. This indicated that further investigation of the effect of deviation from the pre-planned path caused by unknown obstacles would be useful, provided that methods of connecting to those portions of the linked list which were unaffected by the deviation, were determined.

3.3) THE IMPROVEMENTS IN INITIAL PLANNED PATHS IN A SIMPLE KNOWN ENVIRONMENT

Several paths through a known environment consisting of only two isolated rectangular objects were obtained with the aim of indicating quantitatively the improvements (i.e. reduction in path lengths) produced by revisions of the initial planned path, compared with the defined shortest routes. Six paths through the environment were generated and the initial, improved and final paths displayed (see figures 3.37-58). The path lengths were determined and table 3.3 produced.
Figure 3.37 Path 7: the initial path

Figure 3.38 Path 7: the second path

Figure 3.39 Path 7: the third path

Figure 3.40 Path 7: the final path
Finish 3.41 Path 8: the initial path

Figure 3.42 Path 8: the second path

Figure 3.43 Path 8: the third path
Figure 3.45 Path 9: the initial path

Figure 3.46 Path 9: the second path

Figure 3.47 Path 9: the third path

Figure 3.48 Path 9: the final path
Figure 3.49 Path 10: the initial path

Figure 3.50 Path 10: the second path

Figure 3.51 Path 10: the third path

Figure 3.52 Path 10: the final path
Figure 3.57 Path 12: the second path

Figure 3.58 Path 12: the final path
<table>
<thead>
<tr>
<th>Path number</th>
<th>Initial length</th>
<th>Revised length</th>
<th>Shortest path</th>
<th>Improvement achieved (%)</th>
<th>Maximum (%) improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>172</td>
<td>155</td>
<td>131</td>
<td>11</td>
<td>31</td>
</tr>
<tr>
<td>8</td>
<td>166</td>
<td>140</td>
<td>140</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>9</td>
<td>185</td>
<td>142</td>
<td>142</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>10</td>
<td>168</td>
<td>142</td>
<td>142</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>11</td>
<td>162</td>
<td>143</td>
<td>130</td>
<td>13</td>
<td>25</td>
</tr>
<tr>
<td>12</td>
<td>172</td>
<td>157</td>
<td>130</td>
<td>10</td>
<td>32</td>
</tr>
</tbody>
</table>

Table 3.3 Improvements produced by revised paths in simple environment (path lengths (in mm) measured from computer printouts)

<table>
<thead>
<tr>
<th>Path number</th>
<th>Initial length</th>
<th>Revised length</th>
<th>Shortest path</th>
<th>Path taken</th>
<th>Improvement achieved (%)</th>
<th>Maximum improvement (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>198</td>
<td>-</td>
<td>206</td>
<td>298</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
<td>-</td>
<td>90</td>
<td>90</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>259</td>
<td>249</td>
<td>245</td>
<td>286</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>298</td>
<td>217</td>
<td>206</td>
<td>217</td>
<td>37</td>
<td>45</td>
</tr>
<tr>
<td>5</td>
<td>90</td>
<td>-</td>
<td>90</td>
<td>90</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>289</td>
<td>251</td>
<td>251</td>
<td>251</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 3.4 Improvements produced by revised paths (path lengths (in mm) measured from the computer printouts)
Where the obstacle avoidance algorithm made the correct decisions in paths 8, 9 and 10, the maximum possible improvements on the initial paths were obtained, and even for the other paths, 7, 11 and 12, significant improvements were achieved.

The same information was tabulated for the paths through the more complex environments of section 3.2, see table 3.4. The last three paths were not useful as they were obtained when the environment data was inaccurate and the paths found were not used. For paths 1 and 3 in known environments, the results indicated that where significant improvements in the initial path were possible, the revised paths would obtain them. Path 2 did not encounter any obstacles and therefore no improvement was possible.

3.4) CONCLUSIONS

From the results obtained it was concluded that the method of path finding and following with associated obstacle avoidance was a satisfactory method for implementation on a prototype mobile robot for further testing in a physical environment.

It was also concluded that the polygon model of the objects in the environment was suitable for the simulation and that by using data obtained from the simulated rangefinder and bump detectors, paths could be planned through known environments, with revised paths reducing the total length, and through unknown and partially known environments using only the obstacle avoidance algorithm.

It was also concluded that the cartesian grid model was not a satisfactory method of simulating the environment for implementation on the prototype mobile robot, due to the inaccuracy of the representation and subsequent restriction of only being able to move in eight directions. These
difficulties could have been surmounted by reducing the grid size until it was significantly smaller than the area occupied by the mobile robot, but this required a much larger memory capability than was available for the prototype mobile robot. Therefore an alternative was required which increased the accuracy of the environment representation but not the memory requirements.

One aspect of the simulation not considered was the necessity to determine the position of the mobile robot in the environment. This was a pre-requisite for the correct operation of the path following algorithm and was not investigated in the simulation as the position was always assumed to be accurate. Therefore a method of determining the position of the mobile robot in the environment was required.
CHAPTER FOUR

AN IMPROVED MODEL OF THE MOBILE ROBOT

The simulation of the grid environment described in chapter three was considered to be unsuitable for controlling the prototype mobile robot (described in appendix C), due to the inaccuracy of the environment representation. Therefore a similar but more accurate model of the environment was proposed, the quasi-continuous X-Y cartesian coordinate system. This chapter describes the changes required to implement the new coordinate system and discusses the results obtained from the prototype mobile robot. Section 4.1 discusses the new environment model, section 4.2 the simulation of the mobile robot, section 4.3 the path finding implementation including some results, and section 4.4 describes the position location process, also including some preliminary results.

The PASCAL program which implemented the new environment model is shown in appendix B. The programs were developed in their final form on a microprocessor development system [Hewlett-Packard 1981] which extended the standard PASCAL language as defined by Jensen and Wirth [1975], to allow separate compilation of procedures and the inclusion of assembly language programs. To make the best use of these facilities each PASCAL procedure was compiled separately and then linked together before being executed. In this way changes to a procedure did not require the re-compilation of the complete program. Also, if a procedure required a major change then provided the interface to the rest of the program was maintained there was no need to consider how it would affect the rest of the program, and the structure was
maintained. The compiler options used are described at the beginning of appendix B. Normally any references to the program will refer to the listings in appendix B by page number, but occasionally a procedure may be included in the text of this chapter if a particular point is of special interest.

4.1) THE ENVIRONMENT MODEL

With the quasi-continuous coordinate system it was difficult to map the environment in the computer memory directly, instead, the perimeters of the objects were modelled by polygons (see section 2.1.2.1). With this technique the object description only occupied a few memory locations thereby allowing a bigger environment to be modelled at any one time. In addition, it was required that the same method of path finding be used in the simulated environments as in the physical environments, that is, the procedures used to control the simulated mobile robot were also used to control the prototype mobile robot, with only sensor data and the mobile robot movements reflecting which environment was in use. To achieve this a global flag called SIMULATION was defined in PROTO (see page B6) which indicated whether simulated sensor data and movements were required (SIMULATION = TRUE) or prototype sensor and movements (SIMULATION = FALSE). Section 4.1.1 dicuss the implementation of the simulated rangefinder and section 4.1.2 the implementation of the prototype ultrasonic rangefinder. A comparison of the results from the two rangefinders is made in section 4.1.3 and a conclusion drawn in section 4.1.4 as to the suitability of the rangefinders for this mobile robot. The different processes were selected by the user via a Visual Display Unit (VDU) which displayed the message "ENTER COMMAND", and the user then selected which process was required by entering a single character identifier (see PROTO page B6).
4.1.1) Environment data from the simulated rangefinder

The objects were described in the quasi-continuous system by successive coordinate pairs denoting the vertices (or corners) which were then assumed to be joined by straight line segments, thereby defining the obstacle perimeter. The simulation was required to calculate the distance to the nearest obstacle perimeter in the direction of orientation of the robot. The simulated scanning rangefinder and mobile robot were modelled by position and orientation given by \((X,Y,R)\), where \(X\) was the position of the mobile robot along the \(X\)-axis, \(Y\) the position along the \(Y\)-axis and \(R\) the orientation of the mobile robot measured in an anti-clockwise direction from the positive \(X\)-axis. The orientation of the mobile robot and the rangefinder were the same as the rangefinder was physically fixed to the chassis.

The simulated range was calculated by using the simple trigonometric property of the intersection of two straight lines. The two lines were the beam of the rangefinder and the straight line segments formed by the object perimeters. The straight line equations for the rangefinder beam and the side of an object were calculated and the intersection point found. Checks were made to eliminate any object sides parallel to the rangefinder orientation. Then the intersection point was checked to ensure that it was within the corners of the side of the object. A final check was made to determine whether the intersection was "in front" of the rangefinder, as the straight line equation used for the beam of the rangefinder did not distinguish between those intersections "in front" and those "behind". Finally the range was calculated. This process was then repeated for each object side of all the objects as it was not known when a line segment would come within range, even if the corners were apparently out of range. The shortest range obtained to any object side was taken to be the range.
obtained by the rangefinder. It was not necessary to identify which object side was finally used as this data was irrelevant.

A simulated range measurement was implemented by the procedure RANGE_TO_OBJECT (see page B7) which returned the range to the nearest obstacle by repeatedly using the procedure CALC_INTERSECT (see figure 4.1 and page B8), to execute the process described above to calculate the intersection point of every side. The calculation process is outlined below.

Equation of a straight line

\[ Y = MX + C \]

where \( M \) is the gradient of the line and \( C \) the intersection of the line with the \( Y \) axis. The calculations were simplified by subtracting the position of the rangefinder from the corners defining the line segment being tested. The equations of the two straight lines then became:

\[ Y_{\text{object}} = M_{\text{object}}X_{\text{object}} + C_{\text{object}} \quad \cdots (1) \]

\[ Y_{\text{sensor}} = M_{\text{sensor}}X_{\text{sensor}} \quad \cdots (2) \]

At the point of intersection the values of \( X \) and \( Y \) would be the same in both equations, denoted \( X_{\text{intersect}} \) and \( Y_{\text{intersect}} \). Substituting equation (2) into (1) gave:

\[ X_{\text{intersect}} = C_{\text{object}}/(M_{\text{sensor}} - M_{\text{object}}) \quad \cdots (3) \]
PROCEDURE CALC_INTERSECT(VAR RANGE : REAL;
X1,Y1,X2,Y2 : REAL);
BEGIN(*CHECKS TO PREVENT DIVIDE BY ZEROS*)
  WITHIN_SIDE := TRUE; (*INITIALISE*)
  IF ABS(X1-X2) > 0.01 THEN XDFF := TRUE ELSE XDFF := FALSE;
  IF ABS(COS(SENSOR.ANGLE)) > 0.01 THEN NOTZERO := TRUE ELSE NOTZERO := FALSE;
  IF XDFF AND NOTZERO THEN
    (*FIND STRAIGHT LINE EQUATIONS OF LINE SEGMENT AND SENSOR BEAM*)
    BEGIN
      M_OBJECT := (Y1-Y2)/(X1-X2);
      C_OBJECT := Y1 - (M_OBJECT * X1);
      M_SENSOR := SIN(SENSOR.ANGLE)/COS(SENSOR.ANGLE);
      (*NOW SOLVE THE SIMULTANEOUS EQUATIONS*)
      IF ABS(M_SENSOR - M_OBJECT) < 0.01 THEN WITHIN_SIDE :=
        FALSE
      ELSE
        BEGIN
          XINTERSECT := X1;
          YINTERSECT := SIN(SENSOR.ANGLE) * XINTERSECT/COS(SENSOR.ANGLE);
        END
    END
    ELSE (*IF INFINITE LINE SEGMENT GRADIENT USE A DIFFERENT METHOD*)
      IF NOT XDFF AND NOTZERO THEN
        BEGIN
          XINTERSECT := X1;
          YINTERSECT := SIN(SENSOR.ANGLE) * XINTERSECT/COS(SENSOR.ANGLE);
        END
      ELSE (*IF A ZERO SENSOR GRADIENT*)
        IF XDFF AND NOT NOTZERO THEN
          BEGIN
            XINTERSECT := 0.0;
            M_OBJECT := (Y1-Y2)/(X1-X2);
            C_OBJECT := Y1 - (M_OBJECT * X1);
            IF ABS(M_OBJECT) < 0.01 THEN YINTERSECT := Y1
            ELSE YINTERSECT := (XINTERSECT - C_OBJECT)/M_OBJECT;
          END
        ELSE (*IF THIS LINE IS EXECUTED THE TWO LINES ARE EFFECTIVELY PARALLEL*)
          WITHIN_SIDE := FALSE;
        (*NOW PERFORM CHECKS THAT INTERSECTION IS WITHIN THE OBJECT SIDE AND INFRONT OF THE SENSOR, THE X AND Y CHECKS ARE PERFORMED SEPARATELY*)
        CHECK_INTERSECT(XINTERSECT,Y1,Y2,WITHIN_SIDE);
        (*CHECK INFRONT OF SENSOR*)
        INFRONT_CHECK(XINTERSECT,YINTERSECT,WITHIN_SIDE);
        IF WITHIN_SIDE THEN(*CALCULATE THE RANGE TO INTERSECTION*)
          RANGE := -SQRT(XINTERSECT* XINTERSECT) + (YINTERSECT * YINTERSECT))/2;
          ELSE RANGE := MAXRANGE + 1.0;
        END
      END(*CALC_INTERSECT*)
    END(*CALC_INTERSECT*)
  END(*CALC_INTERSECT*)
END;
\[ Y_{\text{intersect}} = \frac{M_{\text{sensor}} C_{\text{object}}}{M_{\text{sensor}} - M_{\text{object}}} \] ......(4)

Checks were made to prevent arithmetical errors caused by division by zero.

The test for intersection within the side of the obstacle was performed by the procedure CHECK_INTERSECT (see page B9) which checked that the intersection value was within the range from one corner point to the next. This was performed separately for each axis as described below.

If \( X_{\text{intersect}} \) was greater than \( X_{\text{vertex1}} \) AND \( X_{\text{vertex2}} \) or smaller than \( X_{\text{vertex1}} \) AND \( X_{\text{vertex2}} \), then the intersection was not within the obstacle side under test.

A similar test was performed on the \( Y_{\text{intersect}} \) point.

The final test to ensure that the intersection point was in front of the rangefinder and not the mirror image behind it was performed by the procedure CHECK_INFRONT (see page B9), by knowing that as the range calculations were performed with respect to the position of the simulated mobile robot the quadrant in which the mobile robot was pointing (i.e. the orientation) was also the one in which the intersection should take place. The quadrant was found from the signs of the \( X \) and \( Y \) coordinates of the intersection point and matched with the orientation. If there was no match then the intersection could not have occurred in front of the simulated mobile robot.

If there were no obstacle sides obstructing the beam of the rangefinder the maximum range was returned. This was taken to be 5 metres for the simulation as the maximum range obtainable from the prototype ultrasonic rangefinder was 5
metres due to signals detected from floor reflections at ranges greater than 5 metres.

The decision to perform a simulated or prototype measurement was taken in the procedure \texttt{ONE\_PULSE} (see figure 4.2 and page B10) depending upon the state of the global flag \texttt{SIMULATION}. When \texttt{SIMULATION} = \texttt{TRUE} the procedure \texttt{RANGE\_TO\_OBJECT} was used to calculate the range value using the simulated environment, otherwise the commands for a range measurement using the ultrasonic rangefinder were issued via the parallel communication bus of the prototype mobile robot (see appendix C).

A scan of the environment was taken to be $180^\circ$, ($-90^\circ$ to $+90^\circ$ with respect to the orientation) and was executed by the procedure \texttt{SCAN\_ENVIRON} (see page B11). The scan was performed by rotating the mobile robot a small fixed angle and taking a range measurement. The resulting range was saved and another angular step taken and the process repeated. The range values were stored in the global array \texttt{RANGE\_ARRAY} defined in \texttt{PROTO}, for use by other functions of the mobile robot; their source, either prototype or simulation, was not identifiable.

4.1.2) Environment data from the prototype ultrasonic rangefinder

As already mentioned above the range value from the ultrasonic rangefinder was initiated via the parallel communication bus. The long range ultrasonic rangefinder controller (see appendix C) then performed the range measurement and returned the range value, in centimetres, to the central processor (the \texttt{SDK-86}) which was executing the \texttt{PASCAL} program. (Note: the ultrasonic rangefinder was the only sensor implemented on the prototype mobile robot as the bump sensor recommended in section 2.4.3 was not installed.)
PROCEDURE ONE_PULSE(VAR RANGE : REAL);

VAR

SEND_MODULE : INTEGER;

BEGIN
  IF SIMULATION THEN
    BEGIN
      SENSOR := ROBOT;
      RANGE_TO_OBJECT(WORLD, NUMBLINES, RANGE);
    END
  ELSE
    BEGIN
      (*INITIATE PULSE*)
      ADDRESS := 4;
      NUMBYTES := 1;
      DSTACK[1] := 1;
      SEND;
      RECEIVE; (*WAIT FOR RANGE*)
      SEND_MODULE := DSTACK[1];
      (*SAVE RANGE IN ARRAY*)
      IF SEND_MODULE = 4 THEN
        BEGIN
          (*CONVERT BYTE TO REAL*)
          BYTE := DSTACK[4];
          SIGNED 8; (*ASSM ROUTINE TO CONVERT TO SIGNED_8*)
          RANGE := I28.0 * OVERBYTE + BYTE;
          RANGE := RANGE + (DSTACK[3] * 256.0);
        (*CONVERT RANGE TO CM*)
        RANGE := RANGE * 2/3;
        RANGE := RANGE + ROFFSET;
      END; (*IF*)
    END;
    WRITELN_REAL(RANGE);
  END; (*ELSE*)
END; (*ONE_PULSE*)

Figure 4.2 The procedure ONE_PULSE
4.1.3) A comparison of simulated and prototype environment data

To determine the accuracy of the environment data from the simulated rangefinder with respect to the prototype rangefinder, a series of environment scans were obtained. The environment scan range data was available in two formats, either as a list of range values using the procedure LIST ARRAY (see page B12), or pictorially using the procedure PRINT SCAN (see page B13) with the ranges (from 0 cm to 500 cm) plotted against the angle turned through (0° to 180°) on an X-Y cartesian coordinate system. For example, the environment shown in figure 4.3 produced the environment scans displayed in figures 4.4, 4.5, and 4.6. The prototype rangefinder environment scans are shown in figures 4.4(a), 4.5(a) and 4.6(a) and the simulated rangefinder scans in figures 4.4(b), 4.5(b) and 4.6(b).

The simulated environment scans clearly identified the objects and even identified the observable corners of the objects. When the environment scans from the prototype ultrasonic rangefinder were considered the objects were not so readily identifiable as environment details were obscured by the ultrasonic beam and objects appeared to be larger than they were. Also the perimeters of the objects were not the straight line approximations shown in figure 4.3 so that the object outlines were not as clear. However, the objects were still identifiable even if the detail was reduced.

There were two factors which caused the difference in simulated and prototype rangefinder environment scans; firstly, the beam spread of the prototype rangefinder, which was not modelled on the simulated rangefinder, made obstacles appear larger than they were and secondly, obstacle surfaces which did not present a normal or near normal surface to the beam of the rangefinder reflected the signal away from the detector and were therefore not
Figure 4.3 The environment used for the rangefinder scans
Figure 4.4(a) Prototype rangefinder: scan 1

Figure 4.4(b) Simulated rangefinder: scan 1
Figure 4.5 (a) Prototype rangefinder: scan 2

Figure 4.5 (b) Simulated rangefinder: scan 2
Figure 4.6 (a) Prototype rangefinder: scan 3

Figure 4.6 (b) Simulated rangefinder: scan 3
detected. This was because at the frequency used (approximately 60 KHz) the reflections were specular instead of the preferable diffuse reflections. Both these effects could have been simulated but this was not considered worthwhile as the ultrasonic rangefinder was not the ideal rangefinder for this purpose. Instead, if improved performance was required an optical rangefinder would be installed which did not suffer from beam spread and specular reflection problems.

4.1.4) Conclusion

At this point the similarity between the simulated and prototype rangefinder scans indicated that the path finding and position location processes could operate satisfactorily using data from this sensor. Although this was confirmed for the path finding and following processes (see section 4.3), it was found that the inaccuracies in the range measurements caused the position location process to produce widely spread results for some configurations of objects (see section 4.4).

4.2) THE SIMULATION OF THE MOVEMENTS OF THE MOBILE ROBOT

In the quasi-continuous cartesian coordinate system in which the prototype mobile robot was to be operated it was necessary to simulate some of the physical characteristics accurately. The minimum requirement was that the orientation of the prototype mobile robot and the position, denoted by \((X,Y,R)\) as discussed in section 4.1.1, be maintained at all times. The minimum movements required to implement the Means-end path finding algorithm were stationary rotation and straight line translation. The acceleration, motoring and deceleration functions of the individual drive motors were controlled by the low level controller (see appendix C) from commands sent via the parallel communication bus.
The restriction of possible movements also reduced the calibration required (see section 4.2.2) as concluded in section 2.3.4.1, as "assumed movement parameters" were used to interpolate the position and orientation of the prototype mobile robot between absolute position fixes.

4.2.1) Simulation of the rotation and translation movements

As assumed movement parameters were used to maintain an accurate representation of the position of the prototype mobile robot the position was always effectively simulated. Therefore the position of the mobile robot had to be updated for every movement made, independent of the state of the flag SIMULATION. When SIMULATION was set TRUE it was assumed that a copy of the original position had been saved in memory before any movement instructions updated the position variable, then when the simulation period was ended (SIMULATION = FALSE) the original position was re-entered into the position variable. The position variable was a global PASCAL record called ROBOT defined in PROTO (see page B6).

When SIMULATION was set FALSE the appropriate movement commands were also sent to the prototype mobile robot, as well as updating the position variable. This process was implemented in the procedure MOVE_ROB (see figure 4.7 and pages B14). The movements were translations forwards or backwards by the specified distance, in millimetres, with the existing orientation, or stationary rotation left (anticlockwise) or right (clockwise) the specified distance, which again was in millimetres and had to be converted to radians. The position variable was updated as follows:-

**Forwards**

\[ x_{\text{robot}} := x_{\text{robot}} + \text{Distance} \times \cos(\text{Orientation}_{\text{robot}}) \]
PROCEDURE MOVE_ROB (MOVEMENT : CHAR;
    REAL DIST : REAL;
    RATIO : INTEGER);

BEGIN
(*UPDATE SIMULATION FIRST*)
CASE MOVEMENT OF
'F' : WITH ROBOT DO
BEGIN
    XCOOR := XCOOR + (REAL_DIST*COS(ANGLE));
    YCOOR := YCOOR + (REAL_DIST*SIN(ANGLE));
END;(*WITH*)
'B' : WITH ROBOT DO
BEGIN
    XCOOR := XCOOR -(REAL_DIST*COS(ANGLE));
    YCOOR := YCOOR -(REAL_DIST*SIN(ANGLE));
END;(*WITH*)
'L' : WITH ROBOT DO
    ANGLE := ANGLE + (REAL_DIST/WHEELBASE);
'R' : WITH ROBOT DO
    ANGLE := ANGLE - (REAL_DIST/WHEELBASE);
END;(*CASE*)
IF ROBOT.ANGLE > TWO_PI THEN
    WITH ROBOT DO
    REPEAT ANGLE := ANGLE - TWO_PI
    UNTIL ANGLE < TWO_PI
ELSE
    IF ROBOT.ANGLE < ZERO THEN
        WITH ROBOT DO
        REPEAT ANGLE := ANGLE + TWO_PI;
        UNTIL ANGLE > ZERO;
    WRITE_REAL(ROBOT.XCOOR);WRITE_REAL(ROBOT.YCOOR);WRITELN_REAL(ROBOT.ANGLE * 1000.0);
    IF NOT SIMULATION THEN
    BEGIN
(*CONVERT DISTANCE TO STEPS*)
    IF RATIO = 1 THEN REAL_STEPS := REAL_DIST * DCONST1
    ELSE IF RATIO = 7 THEN REAL_STEPS := REAL_DIST * DCONST7
    ELSE IF RATIO = 2 THEN REAL_STEPS := REAL_DIST * DCONST2
    ELSE IF RATIO = 3 THEN REAL_STEPS := REAL_DIST * DCONST3
    ELSE IF RATIO = 4 THEN REAL_STEPS := REAL_DIST * DCONST4
    ELSE IF RATIO = 5 THEN REAL_STEPS := REAL_DIST * DCONST5
    ELSE IF RATIO = 6 THEN REAL_STEPS := REAL_DIST * DCONST6
    ELSE WRITELN CHAR('ERR IN MOVE ROB');
    DISTANCE := ROUND(REAL_STEPS);
    DIST_COUNT := 255;
    WHILE DISTANCE > 255 DO
    BEGIN
        DIST_COUNT := DIST_COUNT -1;
        DISTANCE := DISTANCE - 255;
    END;
DISTANCE := 255 - DISTANCE;
    TEMP_INT := 96 + RATIO;
CASE MOVEMENT OF
'F' : TEMP_INT := TEMP_INT + 16;
'B' : TEMP_INT := TEMP_INT + 16 + 8;
'L' : TEMP_INT := TEMP_INT + 8;
'R' : TEMP_INT := TEMP_INT + 0;
OTHERWISE TEMP_INT := TEMP_INT + 16;
END;(*CASE*)
(*SEND TO MOTOR DRIVE CONTROLLER*)
ADDRESS := 1;
NUMBYTES := 3;
DSTACK[1] := TEMP_INT;
DSTACK[2] := DIST_COUNT;
DSTACK[3] := DISTANCE;
SEND;(*TO ROBOT*)
(*WAIT FOR CONFIRMATION OF MOVEMENT*)
RECEIVE;(*CONFIRMS MOVE COMPLETED*)
END;
END;(*MOVE_ROB*)

Figure 4.7 The procedure MOVE_ROB
\[ Y_{\text{robot}} := Y_{\text{robot}} + \text{Distance} \times \sin(\text{Orientation}_{\text{robot}}) \]
\[ \text{Orientation}_{\text{robot}} := \text{Orientation}_{\text{robot}} \]

**Backwards**

\[ X_{\text{robot}} := X_{\text{robot}} - \text{Distance} \times \cos(\text{Orientation}_{\text{robot}}) \]
\[ Y_{\text{robot}} := Y_{\text{robot}} - \text{Distance} \times \sin(\text{Orientation}_{\text{robot}}) \]
\[ \text{Orientation}_{\text{robot}} := \text{Orientation}_{\text{robot}} \]

**Left**

\[ X_{\text{robot}} := X_{\text{robot}} \]
\[ Y_{\text{robot}} := Y_{\text{robot}} \]
\[ \text{Orientation}_{\text{robot}} := \text{Orientation}_{\text{robot}} + \frac{\text{Distance}}{\text{Wheelbase}} \]

**Right**

\[ X_{\text{robot}} := X_{\text{robot}} \]
\[ Y_{\text{robot}} := Y_{\text{robot}} \]
\[ \text{Orientation}_{\text{robot}} := \text{Orientation}_{\text{robot}} - \frac{\text{Distance}}{\text{Wheelbase}} \]

The distance to be rotated, which was the circumference of an arc described by the wheels, was converted to an angle by dividing by the diameter of the circle (i.e. the wheelbase of the mobile robot).

When these movements were to be executed by the prototype mobile robot (i.e. SIMULATION = FALSE) the distance had to be converted into the number of steps to be executed by the drive step motors. Due to the low level control process
implemented on the drive motor controller (see appendix C) the number of steps per millimetre was dependent on the speed required: seven speeds were available and seven conversion constants were required. Also the PASCAL control program waited until the movement was completed before continuing, to prevent the simulation becoming several moves ahead of the mobile robot. This was a restriction on the speed of operation and it was hoped to remove this at some future time.

4.2.2) Calibration of the prototype mobile robot movements

The calibration process was performed to ensure a close correlation between the position of the prototype and simulated mobile robots as required by the "assumed movement parameter" technique used for position interpolation. This reduced the need for absolute position fixes and also enabled the path following process to operate with the minimum of problems.

The calibration process consisted of three tests. First, the straight line translation was calibrated by specifying a linear translation forward of one metre. The constants used to convert from distance to the number of steps required at each speed were then calculated. This process was repeated until the movement executed was within the minimum resolution of the measurement system. The distances were measured using a metre rule and the maximum error was taken to be +/- 2 mm. The procedure MOVE_GET (see page B35) was used to move the prototype mobile robot for the calibration tests.

The second test was to turn through a specified angle, normally chosen to be 90° and check that the correct angle had been turned through. As the linear translations had already been calibrated the rotation distances were also calibrated as the drive wheels were moved in opposite
directions to produce rotations and in the same directions for translations. What was required was to convert from an angle into a distance and vice-versa which was performed using the relationship:

\[
\text{circumference of circle} = \pi \times \text{diameter of circle}
\]

The diameter of the circle was equal to the wheelbase of the chassis which was 34.4 cm. The angle turned through was found to be within the minimum resolution of the measurement system (i.e. +/- 2 mm) when this constant was used to convert from angle to distance.

The final test was performed during one of the path finding experiments (see section 4.3.2) where Path 10 was a continuous series of movements around a single object (see figure 4.17) with a total path length of 1192 cm, and a cumulative angle turned through of 834°. The error of the measured start and finish positions was less than +/- 2.5 cm and the difference of the simulated finish position, which had been updated using the assumed movement parameters, and the measured finish position of the prototype mobile robot, was -5 cm in the X direction and -13 cm in the Y direction. Table 4.1 gives a list of the errors in the X and Y coordinates as the test was proceeding. These values were well within the specified measurement error values and indicated that the calibration process and interpolation of the position by using assumed movement parameters were satisfactory methods of maintaining an accurate representation of the position of the prototype mobile robot. The orientation of the prototype mobile robot was measured at the start position with reference to a fixed axis but there was no accurate method of measuring orientation at the finish position. Therefore no definite conclusion regarding the accuracy of the orientation was made, but it was assumed to be satisfactorily accurate because of the accuracy of the final position measurement.
Table 4.1 The intermediate position errors for Path 10

<table>
<thead>
<tr>
<th>Cumulative Distance (cm)</th>
<th>Angle (°)</th>
<th>Error X-axis (cm)</th>
<th>Error Y-axis (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>306</td>
<td>108</td>
<td>1</td>
<td>-6</td>
</tr>
<tr>
<td>356</td>
<td>178</td>
<td>-6</td>
<td>-6</td>
</tr>
<tr>
<td>676</td>
<td>381</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>776</td>
<td>508</td>
<td>-6</td>
<td>13</td>
</tr>
<tr>
<td>1102</td>
<td>710</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>1192</td>
<td>832</td>
<td>-5</td>
<td>-13</td>
</tr>
</tbody>
</table>
4.3) PATH FINDING AND FOLLOWING WITH THE IMPROVED SIMULATED MOBILE ROBOT

The objective of the path finding and following algorithm was to find a path through the environment using the simulation and then for the prototype mobile robot to follow that path through the physical environment, without encountering any objects. In the initial simulation described in chapter three both these functions were provided by the model and no prototype mobile robot was involved. For implementation on the prototype mobile robot changes were made to the simulation of the rangefinder and movements of the mobile robot (see sections 4.1 and 4.2) in order to closely model the required processes. This also created the necessity of changing the path finding algorithm and in particular the obstacle avoidance algorithm, to take advantage of the information provided by the rangefinder. Path finding in simulated and physical environments is described in section 4.3.1. The path following process was virtually the same as the initial simulation in chapter three except that the PASCAL implementation was improved and this is described in section 4.3.2. Unfortunately it was not possible to implement the path finding process in the simulated environment and then follow the found path in the physical environment due to technical difficulties with the chassis of the prototype mobile robot. Section 4.3.3 comments on the suitability of the techniques implemented so far.

4.3.1) Path finding in the simulated and physical environments

To find an obstacle free path the Means-end algorithm produced the initial path by moving the mobile robot through
the environment, using local obstacle avoidance techniques. This function was implemented by the procedure AVOID (see figure 4.8 and page B16) which used the linked list of destinations and attempted to move in straight lines from one to the next. First the mobile robot orientated with the next destination in the linked list (given by FIRST_POS in the procedure AVOID) by rotating either left or right. Then a range measurement was taken and if the destination was less than the range from the rangefinder the mobile robot was able to move directly to the destination, as there were no obstacles on the straight line path joining them. If there was an object directly in front then the mobile robot moved forward half the distance to the object unless this distance was less than a threshold value (55 cm was used). This moved the mobile robot to within a suitable distance for executing the object avoidance procedures.

The object was negotiated by executing the procedure GET_INTER_DEST (see figure 4.9 and page B17) which select a point at the edge of the object which avoided the object. For the prototype mobile robot it was assumed that all objects were isolated so that when any object was detected there was a "left" and a "right" edge. The edge of the object which caused the least deviation from the present path was chosen as the new direction. A point some distance to one side of the edge chosen was selected which allowed the prototype mobile robot to pass without collision, and was called an Intermediate Destination. The Intermediate Destination was entered into the linked list as the next destination to be visited. (This was the opposite to the previous simulation which generated Intermediate Turning Points which were added to the linked list as positions which had already been visited.)

The edges of the object were detected by the procedure SWEEP_TO_EDGE (see page B18) which rotated the mobile robot and hence the rangefinder, first to the left and then to the
PROCEDURE AVOID(VAR FIRST_POS : POS_PTR;
VAR PATH_CLEAR : BOOLEAN);

VAR

PREV_DEST : POS_PTR;
XDIFF,YDIFF,THETA : REAL;
DISTANCE,RANGE_TO_DEST,ERR_THRES : REAL;
MOVEMENT : CHAR;
TEMP_RANGE : REAL;

BEGIN
ERR_THRES := 0.06; (*3.3 DEGREES*)
PREV_DEST := FIRST_POS;
REPEAT(*UNTIL AT FINAL DESTINATION*)
REPEAT(*UNTIL AT INTER DEST*)
(*ORIENTATE WITH NEXT DEST*)
WITH PREV_DEST,NEXT_POS DO
BEGIN
XDIFF := XPOS - ROBOT.XCOOR;
YDIFF := YPOS - ROBOT.YCOOR;
END; (*WITH*)
THETA := ARCTAN(YDIFF/XDIFF);
(*SELECT CORRECT QUADRANT*)
IF XDIFF < ZERO THEN THETA := THETA + PI
ELSE IF (XDIFF > ZERO) AND (YDIFF < ZERO) THEN
THETA := THETA + TWO_PI;
THETA := THETA - ROBOT.ANGLE;
IF THETA > PI THEN THETA := THETA - TWO_PI
ELSE IF THETA < MINUS_PI THEN THETA := THETA + TWO_PI;
IF ABS(THETA) > ERR_THRES THEN
BEGIN
IF THETA > ZERO THEN MOVEMENT := 'L'
ELSE MOVEMENT := 'R';
DISTANCE := WHEEL_BASE * ABS(THETA);
MOVE_ROB(MOVEMENT,DISTANCE,1);
END ELSE WRITELN_CHAR('LESS THAN THRES');
ONE_PULSE(TEMP_RANGE);
RANGE_TO_DEST := SQR(XDIFF) + SQR(YDIFF);
RANGE_TO_DEST := SQRT(RANGE_TO_DEST);
IF TEMP_RANGE > RANGE_TO_DEST THEN
BEGIN(*IF MORE THAN THRESHOLD DISTANCE MOVE FORWARD*)
DISTANCE := RANGE_TO_DEST;
MOVEMENT := 'F';
MOVE_ROB(MOVEMENT,DISTANCE,1);
ONE_PULSE(TEMP_RANGE);
WITH PREV_DEST,NEXT_POS DO
RANGE_TO_DEST := SQRT(SQR(ROBOT.XCOOR-XPOS)+SQR(ROBOT.YCOOR-YPOS));
END ELSE
BEGIN(*MOVE CLOSER IF NECESSARY*)
DISTANCE := TEMP_RANGE/2.0;
IF DISTANCE > 55.0 THEN
MOVE_ROB('F',DISTANCE,1);(*STOPS IT GETTING TO CLOSE*)
END;(*IF*)
UNTIL RANGE_TO_DEST < 10.0;
PREV_DEST := PREV_DEST,NEXT_POS;
UNTIL PREV_DEST,NEXT_POS = NIL;
(*DISPLAY POSITION*)
WRITE_CHAR('ROBOT AT ');
WRITE_REAL(ROBOT.XCOOR);WRITE_REAL(ROBOT.YCOOR);WRITELN_REAL(ROBOT.ANGLE * 1000.0);
END;(*AVOID*)

Figure 4.8 The procedure AVOID
PROCEDURE GET_INTER_DEST(VAR PREV_DEST : POS_PTR);

VAR

THIS_ANGLE, LEFT_ANGLE, RIGHT_ANGLE : REAL;
LEFT_RANGE, RIGHT_RANGE : REAL;
DEST : POS_PTR;
AV_RANGE, TURN_ANGLE : REAL;

BEGIN(*FIND EDGE WHICH CAUSES THE LEAST DEVIATION*)
THIS_ANGLE := ROBOT.ANGLE;
LEFT_ANGLE := ZERO;
LEFT_RANGE := ZERO;
SWEEP_TO_EDGE('L', LEFT_ANGLE, LEFT_RANGE);
MOVE_ROB('R', LEFT_ANGLE * HALF_WHEELBASE, 1);
RIGHT_ANGLE := ZERO;
RIGHT_RANGE := ZERO;
SWEEP_TO_EDGE('R', RIGHT_ANGLE, RIGHT_RANGE);
IF LEFT_ANGLE < RIGHT_ANGLE THEN
BEGIN(*CALCULATE THE INTERMEDIATE DESTINATION FROM THE EDGE WHICH CAUSES THE LEAST DEVIATION*)
AV_RANGE := LEFT_RANGE;
TURN_ANGLE := LEFT_ANGLE + THIS_ANGLE + (1.15 * ARCTAN(HALF_WHEELBASE / LEFT_RANGE));
(*THIS GIVES A LARGE MARGIN OF AVOIDANCE*)
END
ELSE
BEGIN
AV_RANGE := RIGHT_RANGE;
TURN_ANGLE := THIS_ANGLE - (RIGHT_ANGLE + (1.15 * ARCTAN(HALF_WHEELBASE / RIGHT_RANGE)));
END;
NEW(DEST);
DEST.XPOS := ROBOT.XCOORD + (1.35 * AV_RANGE * COS(TURN_ANGLE));
DEST.YPOS := ROBOT.YCOORD + (1.35 * AV_RANGE * SIN(TURN_ANGLE));
(*INSERT NEW DESTINATION INTO LIST*)
WRITE_CHAR('INTER DEST ');
WRITE_REAL(DEST.XPOS);
WRITE_REAL(DEST.YPOS);
DEST.NEXT_POS := PREV_DEST.NEXT_POS;
PREV_DEST.NEXT_POS := DEST;
END;(*GET_INTER_DEST*)

Figure 4.9 The procedure GET_INTER_DEST
right. A sudden increase in range between two consecutive range measurements indicated that the edge of the object had been scanned. Once the edge which caused the minimum deviation was detected an amount equal to the angle of the arc projected by the width of the prototype mobile robot chassis, plus 15% as a margin of error, was added to the deviation angle. This was the angle to be turned through to avoid the object.

\[
\text{turn angle} = \text{minimum deviation} + 1.15 \times \arctan\left(\frac{\text{width of mobile robot}}{2 \times \text{range to edge}}\right)
\]

Also, it was necessary to move past the edge of the object by a sufficient distance to prevent the mobile robot from backtracking if the destination was still obscured by the same (or a different) object. A suitable amount was taken to be 35% of the range to the selected edge.

The relative position of the Intermediate Destination was then calculated and added to the position of the mobile robot to form the absolute position value, which was then inserted into the linked list of destinations as the next point to be visited. The local flag PATH_CLEAR was set FALSE to indicate that an object had been encountered.

Once the Intermediate Destination had been added to the linked list the simulated mobile robot was orientated with it and attempted to move towards it. Normally no obstacles were encountered but should one be encountered the same process was repeated. At the Intermediate Destination the mobile robot orientated with the original destination and attempted to move towards it. Some examples of paths produced through simple environments are given in section 4.3.3.
This was the method used to find the initial path in both the simulated and physical environments. When the physical environment was used (i.e. SIMULATION = FALSE) the prototype mobile robot arrived at the destination. However, if the simulated mobile robot had been used (i.e. SIMULATION = TRUE) the path, which now contained additional Intermediate Destinations, could be executed again if an object was encountered. If further objects were then encountered on the re-run of the path, more Intermediate Destinations were generated and added to the linked list, and the complete process repeated until the path was negotiated without encountering any objects (i.e. PATH_CLEAR = TRUE).

When an object free path was found from the start to the finish positions the state of the SIMULATION flag was changed to FALSE, and the linked list of start, Intermediate Destinations and finish position, was followed by the prototype mobile robot in the physical environment. Assuming the simulated environment was accurate no objects were encountered. The sequence of events is described by the following pseudo-PASCAL program fragment:-

```
SIMULATION := TRUE;
REPEAT
    initialise position;
    PATH_CLEAR := TRUE;
    AVOID(linked list, PATH_CLEAR);
UNTIL PATH_CLEAR;
SIMULATION := FALSE;
initialise mobile robot position;
AVOID(linked list, PATH_CLEAR);
```
which was implemented in the procedure FIND_PATH (see page B19). (Note: due to technical difficulties the following of the found path in the physical environment was not implemented, and therefore the last two statements do not appear in FIND_PATH.)

However, if an obstacle was encountered which was not present in the model of the environment, the obstacle avoidance technique used to produce the linked list was executed to produce further Intermediate Destinations. Previously planned Intermediate Destinations were not "dumped" as was the case with the grid environment simulation described in chapter three, but a local diversion was generated. This could lead to situations where the prototype mobile robot negotiated an object to reach an Intermediate Destination when it would have been more efficient to proceed to the next position in the linked list, as discussed in section 2.2.2.1. This was an area where further investigation was required.

4.3.2) Examples of paths found

The paths were found using the simulated and prototype mobile robots, for a series of start and finish positions in several simple environments. The results were then compared for accuracy.

The first environment contained four isolated objects in an area of 200 x 450 cm and paths were found for five different start and finish position pairs; the results are shown in figures 4.10 to 4.14. The paths chosen show a satisfactory degree of agreement indicating that the simulated environment was suitable for the path finding process.

The differences in the simulated and physical paths were caused mainly by the beam spread of the ultrasonic rangefinder, which caused objects to appear larger than they
Figure 4.10 Path 1

Figure 4.11 Path 2
Figure 4.12 Path 3

Figure 4.13 Path 4
Figure 4.14 Path 5

Figure 4.15 Paths 6 and 7
actually were. This effect was demonstrated by Path 1 (see figure 4.10) where the prototype mobile robot detected the presence of the small object (at approximately \((120,170)\)) and calculated an Intermediate Destination to avoid it, even though the object was not directly in front when orientated with the destination. The simulated mobile robot did not detect the object and the path taken was more direct. The simulated mobile robot also passed closer to the object than was possible for the prototype mobile robot due to the width of the chassis. Therefore the beam spread of the ultrasonic rangefinder prevented the prototype mobile robot from choosing paths which passed close to objects, and it was planned to implement a similar process in the simulation. The same effect can be seen in Path 2 (see figure 4.11). The remaining three paths had a high degree of agreement, with the same objects detected and almost the same path chosen by the simulated and prototype mobile robots. Differences in paths were due to the beam spread of the ultrasonic rangefinder.

Paths 6, 7, 8 and 9 (see figure 4.15 and 4.16) were found using an environment containing only one object, and demonstrated the same effect, of the prototype mobile robot choosing paths which passed further away from the object than the path chosen by the simulation, as discussed above.

The simulated version of Path 7 appeared to be completely different from the physical path taken, but this was due to the difference in the deviation between turning left or right to move around the object being small, and the difference in range values from the prototype and simulated rangefinders caused a different decision to be made.

Again it was the difference in rangefinders which caused the simulated version of Path 9 (see figure 4.16) to move towards the object before negotiating it, as the threshold distance of 55 cm (see section 4.3.1) was exceeded, whereas
Figure 4.16 Paths 8 and 9

Figure 4.17 Path 10
for the prototype mobile robot this distance was not exceeded and the object was avoided immediately.

Path 10 (see figure 4.17) used the same simple environment and path finding process, to test the calibration of the assumed movement parameters of the prototype mobile robot, (note: this path was not necessary for the simulated mobile robot), as discussed previously in section 4.2.2.

A preliminary solution to negotiating a non-isolated object was implemented in SWEEP_TO_EDGE which aborted the edge detection process if the angle turned through was more than 180\(^\circ\). Greater angles indicated that the mobile robot was probably in some sort of spiral which required additional maze solving techniques (which were not yet implemented) as discussed in section 3.2.1. This technique was suitable for objects with simple "protrusions" as demonstrated by Path 11 (see figure 4.18) but would need to be extended to find paths around objects which formed spirals.

4.3.3) Summary

The PASCAL implementation of the path finding and following Means-end algorithm was explained in detail and related the theory proposed earlier in section 2.2.1.7 to the structure of the program.

The path finding process was then used in a variety of simple environments and the paths generated by the simulated and prototype mobile robots were compared. The differences in the paths were considered to be mainly due to the differences in the range values produced by the simulated and prototype rangefinders. The calibration of the assumed movement parameters used to interpolate the position of the prototype mobile robot between absolute position fixes was also outlined.
PROCEDURE FIND_POSITION;

VAR

PTR, FIRST : TRI_PTR;
NUMB OBJ : INTEGER;
OBJ_LIST : OBJ_ARRAY;

BEGIN

INITHEAP(ADDR(HEAPSTART),HEAPSIZE);
NEW(PTR);
FIRST := PTR;
FIRST.FROM_OBJ := 0; (*ESTABLISHES FIRST - I THINK!*)
DETECT_OBJE(NUMB_OBJ,OBJ_LIST); (*DETECTS ANY OBJECTS*)
GET_KNOWN_OBJ; (*PUTS KNOWN OBJ INTO LINKED LIST*)
IF NUMB_OBJ > 2 THEN
BEGIN
SET_UP_OBJ_DATA(FIRST); (*FORM KNOWN OBJ DATA STRUCTURE*)
OBJECT_MATCH(NUMB_OBJ,OBJ_LIST,FIRST); (*MATCH OBJECT
DISTRIBUTION AND CALCULATION THE VALID POSITIONS*)
END
ELSE WRITELN_CHAR('NOT 3 OBJ');
END; (*FIND_POSITION*)

Figure 4.19 The procedure FIND_POSITION
4.4) FINDING THE POSITION OF THE MOBILE ROBOT

The position location process was the only part of the control software which had not been tested on the grid environment simulation, due mainly to the short range of the rangefinder used. The quasi-continuous environment model was suitable for implementing a much longer range rangefinder and an ultrasonic rangefinder was available with a maximum range of 10 m, as mentioned previously in sections 4.1 and 4.2.

The theory of the technique used to find the position of the mobile robot in the environment using the data available from the rangefinder is discussed in detail in section 2.3.2. The practical PASCAL implementation is described in section 4.4.1 and some preliminary results presented in section 4.4.2.

4.4.1) The implementation of the position finding process

The position finding process consisted of three sequential stages; firstly, a 180° scan of the environment with the rangefinder was made, secondly the positions of the detected objects relative to the rangefinder were obtained from the scan data, and finally the position was calculated when three of the objects had been identified. The process was only of real use with the prototype mobile robot as it was an inherent part of the simulation that the position of the simulated mobile robot was always known. However, the same process could be performed with the simulated mobile robot in order to test the location process in a repeatable manner, by setting SIMULATION = TRUE before the environment data was obtained.

The procedure SCAN_ENVIRON (see page B11) controlled the environment scan and was executed from the main loop of the
The range data was saved in the array **RANGE_ARRAY** defined in the assembly language program **MOB86IOA** (see page B39). The data was obtained from the rangefinder, prototype or simulated, and was available for repeated use (if required) by the rest of the position location process.

Objects were identified by differentiation of the range data by the procedure **DIFF** (see page B20). The identification also included a "noise" removal algorithm which detected spurious or phantom objects which were less than the minimum object width. The smallest object had an apparent size equal to the beam width of the ultrasonic rangefinder and any objects which were less than that were defined as being spurious. (Note: the presence of spurious range values was found to be due to cracks in the floor covering). The unreliable range values were then removed and interpolated values substituted from adjacent measurements. The updated range values were then differentiated again and the results saved in the array **DIFF_ARRAY** (also defined in **MOB86IOA**). The procedure **PRE_PROCESS** (see page B20) implemented this process.

The object edges were defined to be detected in the differentiated range values whenever a threshold value, positive or negative, was exceeded. The procedure **DETECT_EDGES** (see page B21) interrogated the user via the VDU to obtain the required threshold values by using the procedure **GET_THRES** (see page B22). After the threshold values were obtained the range values were differentiated and noise removed by **PRE_PROCESS**, as described above. The detected edges were placed in a list which included the sign of the threshold exceeded (i.e. positive or negative) and passed to the procedure **DETECT_OBJS** (see page B23), which was deemed to have detected an isolated object whenever a positive edge was adjacent to and followed by, a negative edge (see figure 2.10(d)). The range to the middle of the
identified object, the relative angle in the scan and the width of the object were calculated, formed into an object list and passed to the procedure FIND_POSITION (see figure 4.19 and page B24).

Provided that three or more objects were detected the identification process was executed. This process matched the distribution of three detected objects against the distribution of all known object triplets (see section 2.3.2). The known objects were entered into the program at this point by the procedure GET_KNOWN_OBJ (see page B25) and a suitable data structure was constructed by the procedure SET_UP_OBJ_DATA (see page B26) which calculated the distances between all possible object triplets.

For example, assume that there were four known objects, \( O_1, O_2, O_3 \) and \( O_4 \). First the objects were placed into a linked list by the procedure OBJ_DATA (see page B27):

\[
\text{Start} \quad O_1 \ldots O_2 \ldots O_3 \ldots O_4 \ldots \text{NIL}
\]

where NIL denotes an empty pointer and ___ indicated a pointer connection.

Then the distances between all possible object pair combinations were calculated and formed into another linked list using the procedure CALC_DISTS (see page B28):

\[
\text{Start} \quad D_{43} \ldots D_{42} \ldots D_{41} \ldots D_{32} \ldots D_{31} \ldots D_{21} \ldots \text{NIL}
\]

where \( D_{np} \) was the distance between object \( n \) (\( O_n \)) and \( p \) (\( O_p \)). (Note: \( D_{np} = D_{pn} \)).

Each distance, \( D_{np} \), then formed the head of a third linked list containing the distances to all other objects, \( D_{nq} \) and
$D_{pq}$ (where $q \neq n$ or $p$), to form the complete triplet using the procedure SECOND_LEVEL (see page B29) in FIND_POSITION.

The complete data structure formed by the four known objects mentioned above is shown below containing the four possible triplets:

```
Start   D_{43}   D_{42}   D_{41}   D_{32}   D_{31}   D_{21}   NIL
       |       |       |       |       |       |
T_{432} T_{421}   NIL   T_{321}   NIL   NIL   NIL
       |       |       |       |       |
T_{431}   NIL   NIL   NIL   NIL
       |       |
NIL
```

where $T_{npq}$ was a pointer to $D_{npq}$ and $D_{pq}$, which had already been calculated and placed in the previous linked list. For example, the distances between objects 2, 3 and 4 were linked together as shown:

```
...-D_{43}---...
    |       |
T_{432}---D_{42}
    |       |
    |       |
    |       |
    |       |
```

The triplet $T_{npq}$ was equivalent to $T_{pqn}$, or any other combination of the three, because the same three object distances were involved. Only the separation between one pair of objects needed to be included as the head of the linked lists of triplets because if the distance had been incorrect the matching process would have discarded the triplet at a latter stage.
The matching process took each possible triplet of detected but as yet unidentified objects, and calculated the distances between them using the cosine rule (see procedure OBJECT_MATCH page B30) to form $UD_{xy}$, $UD_{yz}$ and $UD_{xz}$, where $UD_{np}$ denoted the distance between the unidentified objects n and p. One of the three distances, normally $UD_{xy}$, was then compared with the distances between all known object pairs. An initial match was considered to have been made when the following inequality was fulfilled:

$$|UD_{xy} - D_{np}| < D_{np}/10$$

that is, the absolute difference in distances between object pair x and y (unidentified) and object pair n and p (known) was less than 10% of the distance between the known objects n and p.

Comparisons were then made with the other values in the triplet chain selected to identify the third object. This level of comparison was performed by the procedure MATCH_SECOND_LEVEL (see page B31). A match was considered to have been made when the second inequality was fulfilled:

$$|UD_{yz} - D_{nq}| < D_{nq}/10$$

To ensure that the correct three objects had been selected the third and final inequality was considered:

$$|UD_{xz} - D_{pq}| < D_{pq}/10$$

and if fulfilled the match was considered to have been made.

Having identified the three detected objects, the position of the mobile robot was calculated using the formula outlined in section 2.3.3 and implemented by the procedure CALC_POS (see page B32).
The matching process then continued until the detected object distributions had been checked against the complete known objects data structure, to ensure that all suitable objects were identified. If more than three objects had been detected from the environment data the process outlined above was repeated for all possible triplets of detected objects. The results of some preliminary identification attempts are shown in section 4.4.2.

From the results in section 4.4.2 it can be seen that sometimes many possible positions were identified by this process. It is possible to reduce the number of possible locations by changing the percentage difference in the inequalities to be less than 5%, however, few if any matches were made if this was implemented. Alternatively, if the percentage of distance difference was increased to 15% then so many matches were made as to render the process useless. The matching process was an area where more consideration was required, possibly by increasing the sophistication of the inequality test to increase the reliability of the results.

4.4.2) Examples of finding the position of the mobile robot

The preliminary results were obtained using small widely spaced objects as these enabled the most accurate measurements of object positions to be obtained from the environment scan. These in turn enabled the most accurate calculations to be made when determining the position of the mobile robot.

The initial environment used is shown in figure 4.20 consisting of five small isolated objects. The prototype mobile robot was placed in the environment and the position location process executed. Forty possible locations were calculated (see figure 4.21), the majority of which were
within a radius of 20cm of the correct location. Subjecting these results to a cluster analysis might have been worthwhile.

When the prototype mobile robot was moved a short distance only ten possible locations were calculated (see figure 4.22), six of which were within a radius of 20 cm of the correct location. The reduced number of locations was caused by the reduction in symmetry of the object distribution as observed from the mobile robot. The matching process was able to discriminate against incorrect triplets because of the variation in distances between the detected objects. Again a cluster analysis would have been useful for this set of results.

Two additional objects were added to the environment (see the top of figure 4.23) to increase the complexity of the object distribution. Of the eleven possible locations calculated, only three were within a radius of 20 cm of the correct location. The increased number of objects reduced the number of correct locations and increased the number of incorrect locations. This was thought to be due to the increased number of known object triplets which enabled more matches to be made, although the incorrect locations were widely spaced. The use of cluster analysis may have detected the few correct results among the many widely spaced incorrect ones.

To further complicate the environment, three more objects were added which were unknown to the prototype mobile robot. This meant that objects were detected from the environment data which could not be matched with known objects and the potential for incorrect locations was increased. The result can be seen in figure 4.24 where only six out of seventeen possible locations were within 20 cm of the correct location. The distribution of incorrect locations was restricted to a narrow segment of the environment and this
Figure 4.20 The initial environment

Figure 4.21 Widely spread objects I

- object  • range measurement

mobile robot at (216,44)

- objects

- calculated positions
Figure 4.22 Widely spread objects II

Figure 4.23 Seven widely spread objects
was considered to be due to the symmetrical distribution of known objects, relative to the prototype mobile robot in that direction, indicating that the symmetry of the distribution of the objects in the environment was possibly a factor in determining the usefulness of this technique.

The prototype mobile robot was moved and the same environment used again (see figure 4.25) where a much more definite cluster was obtained with seven out of nineteen possible locations being within a 20 cm radius of the correct location, which was thought to be due to a reduction in the symmetry of the object distribution as viewed from the prototype mobile robot position.

A scan from a third location was obtained which only had four out of twenty-four possible locations within 20 cm of the correct location (see figure 4.26), due again to the increase in symmetry of object distribution. The four correct locations were considered to form a detectable cluster.

Four of the known objects were transferred to unknown status so that only three of the ten objects were now known. Only two possible locations were calculated (see figure 4.27) both of which were completely wrong. This was not considered surprising because there was only one possible known object triplet, which reduced the possibility of obtaining a correct match. However, this was the first time that no correct locations were obtained, indicating that even with near ideal environment conditions, the correct location may not be identified due to errors in the environment data.

One of the unknown objects was converted back into known object status and the experiment repeated. Two additional possible locations were calculated (see figure 4.28), one of which was correct, indicating that the errors in the range data obtained could be compensated for, to some extent, by
Figure 4.24 Three unknown objects I

- Mobile robot at (250, 25)
- □ - Object known to mobile robot
- ■ - Object previously unknown to mobile robot
- + - Calculated position of mobile robot

Figure 4.25 Three unknown objects II

- Mobile robot at (200, 75)
Figure 4.26 Three unknown objects

Figure 4.27 Seven unknown objects
multiple matches, as there were now four different combinations of known object triplets.

The environment was changed to produce what was considered to be the optimum distribution of objects containing four widely spaced objects (see figure 4.29). The aim was to test the accuracy of the position location process with near ideal environment data. The distant, widely distributed, small objects resulted in only correct locations being calculated, although the accuracy of the correct locations was not high. All three locations were within a radius of 30cm of the correct location. This indicated that with the correct environment data the process could be made to work satisfactorily. The prototype mobile robot was moved to a different location in the same environment and the results shown in figure 4.30 were obtained. The change in position was relatively small but it caused two incorrect locations to be calculated, as well as two correct locations. The spread of results was such that it would be difficult to determine what was the correct position if it was not already known.

The final environment had four objects close to the prototype mobile robot (see figure 4.31) and two very accurate locations were calculated. It had been considered that objects at long ranges would have provided the most accurate results due to the increased distances between them but this result indicated that this may not be completely true.

4.4.3 Summary

The preliminary results obtained indicated that the position location process was partially successful when all objects in the environment were small and isolated. Some trends were tentatively identified; firstly, that increasing the proportion of unknown objects in the environment reduced the
Figure 4.28 Six unknown objects

Figure 4.29 Four widely spaced objects
mobile robot at (375, 0)

Figure 4.30 Four widely spaced objects II

mobile robot at (200, 25)

Figure 4.31 Four near objects
number of correct locations calculated; secondly, that the symmetry of the object distribution relative to the mobile robot affected the number and reliability of calculated positions (although the relationship was not determined with any reliability); and finally, that if several possible locations were calculated then an additional analysis by clustering techniques may increase the reliability and possibly the accuracy of the final position calculation.
CHAPTER FIVE

SUMMARY AND CONCLUSIONS

The objectives outlined in section 1.4 were proposed as a solution to the operation of an untethered mobile robot with four areas of interest being identified; the environment model, path finding and following, position location and environment sensors. The following sections discuss the success of the ideas put forward in the light of the experimental results obtained. Finally there are some recommendations for the continuation of this investigation.

5.1) THE SIMULATED AND PROTOTYPE MOBILE ROBOT

Two different environment models were investigated for the simulated mobile robot. The first, a simple grid square model was used to test the path finding and following techniques proposed in section 2.2.1.7. The simplicity of the grid environment model was useful for the initial stages of development but it was found that the resolution was too low to enable the implementation on a prototype mobile robot. Also, the physical environment was not inherently convertible into a grid environment model, although this could be achieved if large amounts of memory were available.

A second environment model was constructed using the quasi-continuous X-Y cartesian coordinate system and modelled the objects in the environment as polygons composed of straight line segments. This model was suitable for use by the simulated mobile robot and, after some minor modifications to the algorithms, the path finding and following process was successfully implemented, as described in chapter four.
The environment model interfaced with the simulated mobile robot through the simulated rangefinder and simulated movements. Therefore, by providing prototype rangefinder and prototype movements with the same (or almost the same) characteristics, the control process was able to control the prototype, and simulated, mobile robots.

It was therefore concluded that the quasi-continuous X-Y cartesian coordinate system and object modelling process was suitable for the implementation of the control processes (i.e. path finding and position location).

5.2) THE MEANS-END PATH FINDING AND FOLLOWING ALGORITHM

The Means-end path finding algorithm proposed, operated by first finding an initial path through an environment by orientating the simulated mobile robot with the destination, and moving towards it. Whenever an object was encountered local object avoidance techniques were used. Every time an object was negotiated a point was added to the linked list of points which, when joined by straight line segments, defined the path taken. The updated path list was then executed again by the simulated mobile robot if an object was encountered on the initial path. This process was repeated until no objects were encountered. The paths generated in this way were considered to be object free and could then be executed by the prototype mobile robot. Provided that the environment model was accurate the path would be followed without encountering any objects. Due to technical difficulties with the prototype mobile robot controller, paths found by the simulation could not be executed, which meant that the process was not fully tested. However, it was possible to operate the path finding algorithm using the prototype mobile robot and this was used to find paths through physical environments containing isolated objects. This demonstrated that the principle of finding a path using the simulated environment and then
following it in the physical environment was feasible, and that if any objects were encountered the prototype mobile robot would be able to generate a local diversion which avoided the object and linked up with the remainder of the found path.

Therefore, the Means-end algorithm was found to be suitable for finding and following paths in simulated and physical environments. In addition, the same control process could be used, provided that the interface to the environment through the simulated or prototype mobile robots had the same characteristics.

5.3) THE POSITION LOCATION PROCESS

The grid environment was considered unsuitable for testing the position location process outlined in section 2.3 so the preliminary testing was performed using the quasi-continuous coordinate system. The positions were found from environment data provided by scanning the rangefinder (simulated or prototype) through a semi-circle. The range data gathered was then processed to detect the position, relative to the mobile robot, of isolated objects.

The distances between three of the detected objects was matched against the the distances between all possible sets of three known objects. When a match was obtained for all three distances, the three detected objects were considered to have been "recognised", and the object information was used to calculate the position of the mobile robot. Because of the large number of combinations of known object triplets it was inevitable that some incorrect matches would be made and incorrect locations calculated. This was confirmed by the preliminary results which were obtained with favourable distributions of objects in the environment.
Some tentative suggestions were made which related the spread of the incorrect locations and the ratio of correct to incorrect locations, to the symmetry of the object distribution. Also, it appeared that cluster analysis of the set of possible locations would enable the correct position to be identified. This would increase the reliability and accuracy of the final location deemed to be correct.

Therefore, it was considered that the preliminary position location results did not provide sufficiently unambiguous evidence to determine whether or not this technique would be a useful addition to the prototype mobile robot. Possible improvements in the technique were suggested and areas of investigation indicated which it was thought would enable a definite decision to be made.

5.4) CONCLUSION

It was concluded that the quasi-continuous X-Y cartesian coordinate system simulations of the environment, rangefinder and mobile robot were sufficiently accurate to enable path finding techniques to be implemented. Also, that the Means-end path finding and following algorithm was suitable for implementation on a prototype mobile robot using this environment model, with the aim of further developing the prototype into an industrial version.

No definite conclusion was reached regarding the suitability of the position location technique as only a preliminary assessment of the results obtained was made.

5.5) RECOMMENDATIONS

There were two major areas of investigation to be undertaken before a definite decision could be made with regard to producing an untethered mobile robot for use in an industrial environment.
First, the position location process required further investigation to increase the reliability of the results obtained, either by selecting a new technique or improving the existing one.

Secondly, once the position location process was sufficiently accurate and operated correctly when integrated with the remainder of the control process, the investigation could proceed towards constructing an industrial prototype mobile robot to determine whether the ideas proposed in section 1.4 are correct.
REFERENCES AND BIBLIOGRAPHY


AGV-2, 2nd International Conference on Automated Guided Vehicle Systems.


to HILARE". Advance papers of the Sixth International Joint Conference on Artificial Intelligence, Tokyo, Japan, August 20-23, 1979, pp335-337.


Larcombe M. H. E., "Mobile Robots for Industrial Use". The Industrial Robot, Vol 6 (2), June 1979, pp70-76.


Lear-Siegler., ADM-3A+, Dumb Terminal Video Display Unit: Users Reference Manual, Leir-Siegler Inc, Data Products Division, 714 North Brookhurst Street, Anaheim, California, USA.


Moravec H. P., "Visual Mapping by a Robot Rover". Advance papers of the Sixth International Joint Conference on Artificial Intelligence, Tokyo, Japan, August 20-23, 1979, pp598-600.


Seals R. C., "Distributed processing for a Mobile Robot". Microprocessor Workshop, The University of Liverpool, Liverpool, UK, September 6-12, 1982.


Winston P. H., "Children can Teach Mechanical turtles to Migrate". Artificial Intelligence, Addison-Wesley, 1975, pp246-252.
PUBLICATIONS

Seals R. C., "Distributed processing for a Mobile Robot". Microprocessor Workshop, The University of Liverpool, Liverpool, UK, September 6-12, 1982.


Appendix A : Grid Environment Program Listing

* Hull V-mode Pascal Compiler Version 3.2 Run on: 21MAY82
* Username: ELE022
* Filename: ROBTEXT

1 0 0 (*REVISED VERSION 1.3 28 JUNE 1981.*)
2 0 0 (*SIMULATES A ROBOT MOVING IN ENVIRONMENTS USING AN *)
3 0 0 (* X-Y COORDINATE SYSTEM. DESTINATIONS.OBJECTS ARE AVOIDED*)
4 0 0 (* BY GOING ROUND THEIR PERIMETER *)
5 0 0 (* THE LASER RANGEFINDER DOES NOT SEE THROUGH OBJECTS *)
6 0 0
7 0 0 PROGRAM SEARCHARRAY(INPUT,OUTPUT,RESPFILE,TURN2,ENVIRON);
8 0 0
9 0 0 CONST N UMPLACES =3;
10 0 0
11 0 0 TYPE
12 0 0 TYPEOFDECISION = (PRIMARY,SECONDARY,ALTERNATE);
13 0 0 TRIALS = (XX,ONE,TWO,THREE,FOUR,FIVE,SEVEN);
14 0 0 OPTIONS1 = (WHITETOSCREEN,NOTWRITE);
15 0 0 OPTIONS2 = (ENABLE,DISABLE);
16 0 0 TIPSORTURN = (INTERMEDIATE,FINAL);
17 0 1 TURNINFO = RECORD
18 0 1 XPOSITION : INTEGER;
19 0 1 YPOSITION : INTEGER;
20 0 1 ORDER : INTEGER;
21 0 1 CLASSOFFSET : TYPEOFTURN;
22 0 1 END;
23 0 1 TURNPOINTER = "POINTOFTURN;"
24 0 1 POINTOFTURN = RECORD
25 0 1 POSITION : TURNINFO;
26 0 1 NEXT : TURNPOINTER;
27 0 1 END;
28 0 1
29 0 1 POSITION = ARRAY[1..2,1..N UMPLEAC ES] OF INTEGER;
30 0 1 ASCII = 0..255;
31 0 0 CODEARRAY= ARRAY[1..128,1..88] OF CHAR;
32 0 0 MOVEMENT = (NONE,NORTH,SOUTH,WEST,EAST,NWEST,NEAST,SWES:
33 0 0 STAC K = ARRAY[1..5] OF MOVEMENT ;
34 0 0
35 0 0 VAR
36 0 0 TRIALNUMB : TRIALS;
37 0 0 DUMPE D : BOOLEAN;
38 0 0 L,M : INTEGER;
39 0 0 OPTIONS2 : (ENABLE,DISABLE);
40 0 0 SAME : BOOLEAN;
41 0 0 N UMPL E F INES : INTEGER;
42 0 0
43 0 0
44 0 0
45 0 0
46 0 0
47 0 0
48 0 0
49 0 0
50 0 0
51 0 0
52 0 0
53 0 0
54 0 0
55 0 0
56 0 0
57 0 0
58 0 0
59 0 0
60 0 0
61 0 0
62 0 0
63 0 0
64 0 0
65 0 0
66 0 0
67 0 0
68 0 0
69 0 0
70 0 0
71 0 0
72 0 0
73 0 0
74 0 0
75 0 0
76 0 0
77 0 0
78 0 0
79 0 0
80 0 0
81 0 0
82 0 0
83 0 0
84 0 0
85 0 0
86 0 0
87 0 0
88 0 0
89 0 0
90 0 0
91 0 0
92 0 0
93 0 0
94 0 0
95 0 0
96 0 0
97 0 0
98 0 0
99 0 0
100 0 0
101 0 0
102 0 0
103 0 0
104 0 0
105 0 0
106 0 0
107 0 0
108 0 0
109 0 0
110 0 0
111 0 0
112 0 0
113 0 0
114 0 0
115 0 0
116 0 0
117 0 0
118 0 0
119 0 0
120 0 0
121 0 0
122 0 0
123 0 0
124 0 0
125 0 0
126 0 0
127 0 0
128 0 0

Program (NONE,NORTH,SOUTH,WEsto,ROBOT HOving

ASCII CODEARRAY = ARRAY[1..255];

MOVEHENT

AXIS

DIRECTION

TOARRAY := CODEARRAY;

COORDIN := INTEGER;

OBJECT := INTEGER;

STACKPOINTER := INTEGER;

X,Y,C := INTEGER;

STROKED : BOOLEAN;

TOARRAY := CODEARRAY;

COPY (VAR FROM : CODEARRAY);

BEGIN

END A1

Listing Filename:

IN

ROBTEXT

21MAY82

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

PROCEDURE ADDOBJECTS(X,Y,LENGTH,WIDTH:INTEGER);
BEGIN
  FOR I := 1 TO L DO
    FOR J := 1 TO M DO
      1 := TEMPARRAY[I,J] := ' ';
  98 1 END;(*CLEAR*)
  139 0
  0 0 PROCEDURE INITIALISE;
  18 0
  19 0 BEGIN
  20 0 1 NEW(Destination);
  21 1 1 STACKPOINTER := 1;
  22 15 1 DIRECTION(STACKPOINTER) := NONE;
  23 38 1 FOR J := 1 TO M DO
  24 54 1 FOR I := 1 TO L DO
  25 72 2 BEGIN
    74 2 CODEARRAY[I,J] := ' ';
    77 2 ROBOTARRAY[I,J] := ' ';
    180 2 OBJARRAY[I,J] := ' ';
    203 2 END;
  30 221 1 XCOORDIN := 8;
  31 225 1 YCOORDIN := 8;
  32 229 1 FOR J := XCOORDIN-3 TO XCOORDIN +3 DO
  33 242 2 BEGIN
    251 2 FOR I := XCOORDIN-3 TO XCOORDIN +3 DO
      271 2 ROBOTARRAY[I,J] := CODEARRAY[I,J];
    372 2 END;
  381 1 ROBOTARRAY[XCOORDIN,YCOORDIN] := 'X';
  424 1 END;(*INITIALISE*)
  494 0
  0 0 PROCEDURE ADDOBJECTS(X,X,LENGTH,WIDTH:INTEGER);
  41 0 0 (*SPECIFIES TOP LEFT HAND COORDIN OF OBJECT,LENGTH*);
  42 0 0 (* AND WIDTH *)
  43 0
  44 0 1 BEGIN
  45 0 1 X := X +4;
  46 7 1 Y := Y +4;
  47 13 1 FOR J := Y TO Y+WIDTH-1 DO
  48 33 1 FOR I := X TO X +LENGTH-1 DO
  49 55 2 BEGIN
    57 2 CODEARRAY[I,J] := '1';
    100 2 OBJARRAY[I,J] := 'a';
  52 183 2 END;
  161 1 END;(*ADOBJECTS*)
  205 0
  55 0 0 PROCEDURE STARTPLACE(XPOINT,YPOINT:INTEGER);
  56 0 0
  57 0 0 VAR
  58 0 0
  59 0 0 START := TURINFO;
  160 0 0 STARTPOINTER := TURNPOINTER;
  161 0 0
  162 0 1 BEGIN
  163 0 1 NEWSTARTPOINTER;
  164 11 1 START.XPOsITION := XPOINT;
  165 15 1 START.YPOsITION := YPOINT;
  166 19 1 START.ORDER := 1;
  167 23 1 START.CLASSOFTURN := FINAL;
  168 25 1 STARTPOINTER. POSITION := START;
  169 42 1 STARTPOINTER . NEXT := NIL;
  170 55 1 LISTHEAD := STARTPOINTER;
  171 67 1 LASTPLACE := LISTHEAD;
  172 75 1 NEXTPLACE := LASTPLACE;
  173 91 1 END;(*STARTPLACE*)
  174 130 0
  175 0 0 PROCEDURE ENTERPOINTER(XPOINT,YPOINT,SEQUENCE:INTEGER);
  176 0 0
  177 0 0 VAR
  178 0 0
  179 0 0 TEMPOINTER := TURNPOINTER;
  180 0 0 TURINFO := TURINFO;
  181 0 0 SEARCHING := BOOLEAN;
  182 0 0
  183 0 1 BEGIN
  184 0 1 TEMPOINTER.XPOsITION := XPOINT;
  185 5 1 TEMPOINTER.YPOsITION := YPOINT;
  186 9 1 TEMPOINTER.ORDER := SEQUENCE;
  187 13 1 TEMPOINTER.CLASSOFTURN := FINAL;
  188 15 1 IF SEQUENCE > 1 THEN
  189 24 2 BEGIN
  190 24 2 SEARCHING := TRUE;
  191 26 2 NEXTPLACE := LISTHEAD . NEXT;
  192 46 2 TEMPOINTER := TEMPOINTER;
  193 56 2 TEMPOINTER . POSITION := TEMPOINTER;
  194 73 2 IF NEXTPLACE <> NIL THEN
  195 73 2 IF (NEXTPLACE . POSITION . ORDER > SEQUENCE) AND
  196 80 2 (LASTPLACE . POSITION . ORDER < SEQUENCE) THEN
  197 89 2 BEGIN
  198 114 3 LASTPLACE . NEXT := TEMPOINTER;
  199 114 3 TEMPOINTER . NEXT := NEXTPLACE;
  200 136 3 LASTPLACE := LISTHEAD;
  201 158 3 NEXTPLACE := LISTHEAD;
  202 170 3 SEARCHING := TRUE;
  203 182 3 IF NEXTPLACE <> NIL THEN
  204 184 3 IF (NEXTPLACE . POSITION . ORDER > SEQUENCE) AND
  205 186 3 (LASTPLACE . POSITION . ORDER < SEQUENCE) THEN
  206 186 3 BEGIN
  207 186 3 LASTPLACE . NEXT := TEMPOINTER;
  208 188 3 TEMPOINTER . NEXT := NEXTPLACE;
  209 200 3 NEXTPLACE := NEXTPLACE . NEXT;
  210 202 3 ELSE
  211 222 3 BEGIN
  212 222 3 LASTPLACE . NEXT := TEMPOINTER;
  213 222 3 TEMPOINTER . NEXT := NEXTPLACE;
  214 244 3 NEXTPLACE := LISTHEAD;
```haskell
?17 290 3 SEARCHING := FALSE;
?10 292 3 END;
?20 296 2 END;
?21 298 2 ELSE WRITELN('ERROR IN', 'TURN LIST');
?22 333 1 END; (*ENTERPOINT*)
?23 378 0
?24 0 0 PROCEDURE ADDPLACES(PRIORITY, TEMPX, TEMPY : INTEGER);
?26 0 0
?27 0 1 BEGIN
?28 1 TEMPX := TEMPX + 4;
?29 7 1 TEMPY := TEMPY + 4;
?30 13 1 IF PRIORITY = 1 THEN STARTPLACE(TEMPX, TEMPY);
?31 40 1 ELSE ENTERPOINT(TEMPX, TEMPY, PRIORITY);
?32 62 1 OBJARRAY[TEMPX, TEMPY] := '+';
?33 105 1 END; (*ADDPLACES*)
?34 142 0
?35 0 0 PROCEDURE TEST1;
?36 0 0
?37 0 1 WRITE(TURNS, DESTINATION~.POSITION.XPOSITION:3, DESTINATION~.POSITION.YPOSITION:3, DESTINATION~.POSITION.ORDER:3);
?38 21 1 DESTINATION~.POSITION.XPOSITION:3, DESTINATION~.POSITION.YPOSITION:3, DESTINATION~.POSITION.ORDER:3);
?39 41 1 IF DISPLAY NOTWRITE THEN
?40 61 1 WRITE(DESTINATION~.POSITION.XPOSITION:3, DESTINATION~.POSITION.YPOSITION:3, DESTINATION~.POSITION.ORDER:3);
?41 67 1 WRITE(DESTINATION~.POSITION.XPOSITION:3, DESTINATION~.POSITION.YPOSITION:3, DESTINATION~.POSITION.ORDER:3);
?42 87 1 DESTINATION~.POSITION.XPOSITION:3, DESTINATION~.POSITION.YPOSITION:3, DESTINATION~.POSITION.ORDER:3);
?43 107 1 OBJECT[TEMPX, TEMPY] := '+';
?44 127 1 WRITE(TURNS, XXXORDI[3], YCOORDI[3]);
?45 151 1 IF DISPLAY NOTWRITE THEN
?46 157 1 WRITE(XCOORDI[3], YCOORDI[3]);
?47 161 1 IF LASTPLACE <> NIL THEN
?48 188 2 BEGIN
?49 198 2 WRITE(TURNS, LASTPLACE~.POSITION.XPOSITION:3, LASTPLACE~.POSITION.YPOSITION:3);
?50 208 2 LASTPLACE~.POSITION.XPOSITION:3, LASTPLACE~.POSITION.YPOSITION:3);
?51 228 2 IF DISPLAY NOTWRITE THEN
?52 234 2 WRITE(LASTPLACE~.POSITION.XPOSITION:3, LASTPLACE~.POSITION.YPOSITION:3);
?53 254 2 WRITE(LASTPLACE~.POSITION.XPOSITION:3, LASTPLACE~.POSITION.YPOSITION:3);
?54 274 2 WRITE(LASTPLACE~.POSITION.XPOSITION:3, LASTPLACE~.POSITION.YPOSITION:3);
?55 294 2 IF DISPLAY NOTWRITE THEN
?56 304 2 WRITE(LASTPLACE~.POSITION.XPOSITION:3, LASTPLACE~.POSITION.YPOSITION:3);
?57 320 2 END;
?58 322 1 ELSE
?59 322 1 BEGIN
?60 322 2 WRITE(TURNS, ', NIL ');
?61 341 2 IF DISPLAY NOTWRITE THEN
?62 347 2 WRITE(' NIL');
?63 362 2 END;
?64 362 1 IF NEXTPLACE <> NIL THEN
?65 369 2 BEGIN
?66 369 2 WRITE(TURNS, NEXTPLACE~.POSITION.XPOSITION:3, NEXTPLACE~.POSITION.YPOSITION:3);
?67 389 2 WRITE(NEXTPLACE~.POSITION.XPOSITION:3, NEXTPLACE~.POSITION.YPOSITION:3);
?68 409 2 IF DISPLAY NOTWRITE THEN
?69 415 2 WRITE(NEXTPLACE~.POSITION.XPOSITION:3, NEXTPLACE~.POSITION.YPOSITION:3);
?70 435 2 WRITE(TURNS, NEXTPLACE~.POSITION.XPOSITION:3, NEXTPLACE~.POSITION.YPOSITION:3);
?71 455 2 WRITE(TURNS, NEXTPLACE~.POSITION.XPOSITION:3, NEXTPLACE~.POSITION.YPOSITION:3);
?72 475 2 IF DISPLAY NOTWRITE THEN
?73 481 2 WRITE(NEXTPLACE~.POSITION.XPOSITION:3, NEXTPLACE~.POSITION.YPOSITION:3);
?74 501 2 WRITE(TURNS);
?75 505 2 IF DISPLAY NOTWRITE THEN
?76 511 2 WRITE(TURNS, 'WIL');
?77 515 2 END;
?78 517 1 ELSE
?79 517 1 BEGIN
?80 517 1 WRITE(TURNS, 'WIL');
?81 534 2 IF DISPLAY NOTWRITE THEN
?82 540 2 WRITE(TURNS, 'WIL');
?83 557 2 END;
?84 557 1 END; (*TEST*)
?85 606 0
?86 0 0 PROCEDURE COMPARE;
?87 288 0 1 BEGIN
?88 289 0 1 IF (XCOORDI=XOBJECT) AND (YCOORDI=YOBJECT) THEN
?89 290 19 2 BEGIN
?90 291 19 2 LASTPLACE := DESTINATION;
?91 292 31 2 DESTINATION := NEXTPLACE;
?92 293 43 2 IF NEXTPLACE <> NIL THEN
?93 294 50 2 NEXTPLACE := NEXTPLACE~.NEXT;
?94 295 72 2 ELSE
?95 296 72 2 NEXTPLACE := NEXTPLACE~.NEXT;
?96 297 84 2 END;
?97 298 84 1 END; (*COMPARE*)
?98 299 125 0
?99 300 0 0 PROCEDURE PRINTASC(A:ASCII);(*EXTERNAL PROCEDURE*)
?100 301 0 0
?101 302 0 0 PROCEDURE PRINTASC(A:ASCII);(*EXTERNAL PROCEDURE*)
?102 303 0 0 EXTERN;
?103 304 0 0
?104 305 0 1 BEGIN
?105 306 0 1 BEGIN
?106 307 5 1 END; (*PRINTASC*)
?107 308 30 1 BEGIN
?108 309 0 0 BEGIN
?109 310 0 0 BEGIN
?110 311 0 0 BEGIN
?111 312 0 0 BEGIN
?112 313 0 0 BEGIN
?113 314 0 0 BEGIN
?114 315 0 1 BEGIN
?115 316 0 1 BEGIN
?116 317 0 1 BEGIN
?117 318 4 1 BEGIN
?118 319 22 1 BEGIN
?120 320 40 1 BEGIN
?121 321 60 1 BEGIN
?122 322 80 1 BEGIN
?123 323 160 1 BEGIN
?124 324 220 1 BEGIN
?125 325 320 1 BEGIN
?126 326 0 0 BEGIN
?127 327 0 0 BEGIN
```
BEGIN
BEGIN
BEGIN
BEGIN
CASE VERTICE OF
CASE VERTICE OF
CASE VERTICE OF

ELSE
ELSE
ELSE

CASE VERTICE OF
CASE VERTICE OF
CASE VERTICE OF

CASE VERTICE OF
CASE VERTICE OF
CASE VERTICE OF

CASE VERTICE OF
CASE VERTICE OF
CASE VERTICE OF

YOU ARE IN AREA
YOU ARE IN AREA
YOU ARE IN AREA

<p>...more code...<br><br>END;(*CASE*)</p>
664 160 2 FOR I := XCOORDIN - 3 TO XCOORDIN + 3 DO
665 160 3 BEGIN
666 182 3 IF FROMARRAY <> FINALARRAY THEN
667 208 3 CASE RESMATRIX[I,J] OF
668 251 3 '1', '2', '3', '4', '5', RESMATRIX[I,J] := TOARRAY[I,J];
669 333 3 OTHERWISE 333 END;(*CASE*)
670 345 3 IF FINALARRAY[I,J] <> TOARRAY[I,J] THEN
671 368 3 BEGIN
672 457 4 IF TOARRAY[I,J] = '1' THEN
673 503 4 IF FINALARRAY[I,J] = '1' THEN
674 551 4 NEW := FALSE
675 555 4 ELSE BEGIN
676 555 5 NEW := TRUE;
677 555 6 FINALARRAY[I,J] := TOARRAY[I,J];
678 649 6 END;
679 649 7 IF (OPTION1 = WRITETOSCREEN) AND NEW THEN
680 649 8 BEGIN
681 657 9 GOTOXY(J-4,I-4);
682 657 10 WRITE(FINALARRAY[I,J]);
683 679 11 END;
684 739 12 END;
685 749 13 END;
686 749 14 END;
687 749 15 IF OPTION1 = WRITETOSCREEN THEN
688 759 16 BEGIN
689 766 17 GOTOXY(YCOORDIN-4,XCOORDIN-4);
690 766 18 WRITE('X');
691 788 19 TRIALCHAR;
692 796 20 BEGIN
693 800 21 END;
694 800 22 END;
695 800 23 TRIALCHAR;
696 802 24 IF DESTINATION = NIL THEN STOP := TRUE;
697 806 25 IF REPLACE THEN
698 816 26 BEGIN
699 820 27 GOTOXY(TEMP3,TEMP2);
700 820 28 WRITE('**');
701 838 29 END;
702 846 30 END;(*UPDATE*)
703 846 31 END;
704 922 0 PROCEDURE NEGXAXIS;
705 960 0 PROCEDURE POSXAXIS;
706 000 0 THEN XFIRST(PRIMARY)
IF (COORDINATE=OBJECT) AND (YCOORDIN=YOBJECT) THEN

776 125 2 DIRECTION[STACKPOINTER] := NONE;
777 146 2 END;
778 148 1 END; (*WHICH*)
779 210 0
780 0 0 PROCEDURE NEXTMOVE;
781 0 0
782 0 0 VAR
783 0 0
784 0 0 TEMPREAL : REAL;
785 0 0 FINISH : BOOLEAN;
786 0 0
87 0 1 BEGIN
88 0 1 IF FIRSTIME THEN
89 5 2 BEGIN
90 5 2 IF ABS(XCOORDIN-XOBJECT)= ABS(YCOORDIN-YOBJECT) THEN
91 26 3 SMALLDIFF := ABS(XCOORDIN-XOBJECT);
92 26 3 LARGEDIFF := ABS(YCOORDIN-YOBJECT);
93 34 3 END;
94 42 3 ELSE BEGIN
95 44 3 SMALLDIFF := ABS(YCOORDIN-YOBJECT);
96 44 3 LARGEDIFF := ABS(XCOORDIN-XOBJECT);
97 53 3 END;
98 60 3 IF SMALLDIFF = 0 THEN
99 60 2 INTERVAL := LARGEDIFF
100 65 2 ELSE BEGIN
101 71 3 TEMPREAL := LARGEDIFF/SMALLDIFF;
102 71 3 INTERVAL := TRUNC(TEMPREAL);
103 97 3 END;
104 102 3 END;
105 102 2 NUMBAXIS := LARGEDIFF - SMALLDIFF;
106 108 2 FIRSTIME := FALSE;
107 111 2 END;
108 111 1 IF SMALLDIFF = 0 THEN
109 116 2 BEGIN
110 116 2 DIRECTION[STACKPOINTER] := AXIS;
111 148 2 COUNTOTAL := COUNTOTAL + 1;
112 154 2 END;
113 156 1 ELSE
114 156 2 BEGIN
115 156 2 IF (COUNTINT < INTERVAL) AND (NOT AXISEND) THEN
116 170 2 BEGIN
117 170 2 DIRECTION[STACKPOINTER] := AXIS;
118 203 2 COUNTINT := COUNTINT + 1;
119 208 3 COUNTAXIS := COUNTAXIS + 1;
120 214 3 COUNTOTAL := COUNTOTAL + 1;
121 220 3 END;
122 222 2 ELSE
123 222 3 BEGIN
124 222 3 DIRECTION[STACKPOINTER] := DIAGONAL;
125 254 3 COUNTINT := 0;
126 257 3 COUNTOTAL := COUNTOTAL + 1;
127 263 3 END;
128 265 1 IF COUNTAXIS := NUMBAXIS THEN
129 265 1 IF (COORDINATE=DESTINATION).POSITION.XPOSITION
130 272 1 THEN ARRIVE := TRUE;
131 275 1 IF (COORDINATE=DESTINATION).POSITION.YPOSITION
132 279 1 THEN ARRIVE := TRUE;
133 320 1 END; (*NEXTMOVE*)
134 392 0
135 0 0 PROCEDURE INQUIRE (VAR DIRECTION2:MOVEMENT);
136 36 0 0
137 0 0 VAR
138 0 0
839 0 0 TEMPX,TEMPY : INTEGER;
840 0 0
841 0 1 BEGIN
842 0 1 TEST1;
843 7 1 OBSTRUCT := FALSE ;
844 10 1 TEMPX := XCOORDIN;
845 14 1 TEMPY := YCOORDIN;
846 18 1 CASE DIRECTION2 OF
847 26 1 NONE ::
848 28 1 NORTH := TEMPY := TEMPY - 1;
849 36 1 SOUTH := TEMPY := TEMPY + 1;
850 44 1 WEST := TEMPY := TEMPY - 1;
851 52 1 EAST := TEMPY := TEMPY + 1;
852 60 2 NEAST := BEGIN
853 60 2 TEMPX := TEMPX + 1;
854 60 2 TEMPY := TEMPY - 1;
855 72 2 END;
856 74 2 NWEST := BEGIN
857 74 2 TEMPX := TEMPX - 1;
858 80 2 TEMPY := TEMPY - 1;
859 86 2 END;
860 88 2 SEAST := BEGIN
861 88 2 TEMPX := TEMPX + 1;
862 94 2 TEMPY := TEMPY + 1;
863 100 2 END;
864 102 2 SWEAST := BEGIN
865 102 2 TEMPX := TEMPX - 1;
866 108 2 TEMPY := TEMPY + 1;
867 114 2 END;
868 116 2 END; (*CASE*)
869 132 1 TEMPHAR := TOARRAY[TEMPX,TEMPY];
870 153 1 CASE TEMPHAR OF
871 183 1 "O" : OBSTRUCT := FALSE;
872 192 1 "1" : OBSTRUCT := TRUE ;
873 197 2 ** : BEGIN
874 197 2 WRITE('ROBOT');
875 211 2 STOP0 := TRUE;
876 214 2 END;
877 216 1 "1" ;
878 218 1 "*" ;
879 220 1 "*" ;
880 222 1 "*" ;
881 224 1 "X" ;
891 125 1 IF NEXTPLACE <> NIL THEN
902 10 1 IF DESTINATION^ POSITION.CLASSOFTURN = BRANCHES 1
903 10 1 IF DISPLAY = NOTWRITE THEN
904 6 2 BEGIN
905 10 2 TEMPTURN.setPosition := YCOORDIN;
906 10 2 TEMPTURN.POSITION := XCOORDIN;
907 14 2 TEMPTURN.ORDER := LASTPLACE^ POSITION.ORDER;
908 26 2 TEMPTURN.CLASSOFTURN := INTERMEDIATE;
909 28 2 RESMATRIX[XCOORDIN,YCOORDIN] := 1^;I^;
910 71 2 INSERTTURN(TEMPTURN);
911 77 2 OBJARRAY[XCOORDIN,YCOORDIN] := 1^;*
912 120 2 WRITELN(TURNS,TEMPTURN.POSITION,
913 150 2 IF DISPLAY = NOTWRITE THEN WRITELN(TEMPTURN.POSITION,
914 156 2 WRITELN(TEMPTURN.ORDER);
915 206 2 END;
916 206 1 END;(*SHORTROUTE*)
917 259 0
918 0 0 PROCEDURE LISTTURNS;
919 0 0
920 0 0 VAR
921 0 0
922 0 0 POINTER : TURNPOINTER;
923 0 0 ENDLIST : BOOLEAN;
924 0 0
925 0 1 BEGIN
926 0 1 ENDLIST := FALSE;
927 3 1 POINTER := LISTHEAD;
928 15 1 REPEAT
929 15 1 WRITELN(TURNS,POINTER^ POSITION.XPOSITION,
930 35 1 POINT.PART POSITION.YPOSITION,
931 55 1 POINTER^ POSITION.ORDER;
932 79 1 IF DISPLAY = NOTWRITE THEN
933 85 1 WRITELN(POINTER^ POSITION.XPOSITION,
934 105 1 POINTER^ POSITION.YPOSITION,
935 125 1 POINTER^ POSITION.ORDER);
936 180 1 IF POINTER^ NEXT <> NIL THEN
937 164 1 BEGIN
938 164 1 IF POINTER^ NEXT <> NIL THEN
939 186 1 ENDLIST := TRUE;
940 188 1 UNTIL ENDLIST;
941 191 1 END;(*LISTTURNS*)
942 228 0
943 0 0 PROCEDURE DUMP(BRANCHES : TYPEOFTURN);
944 0 0 VAR
945 0 0
946 0 0 NOMORE : BOOLEAN;
947 0 0
948 0 0
949 0 1 BEGIN
950 0 1 NOMORE := FALSE;
951 3 1 IF NEXTPLACE <> NIL THEN
952 10 1 IF NEXTPLACE <> NIL THEN
953 10 1 IF DESTINATION^ POSITION.CLASSOFTURN = BRANCHES 1
954 25 2 BEGIN
955 25 2 BEGIN
956 37 2 LASTPLACE^ NEXT := NEXTPLACE;
957 59 2 NEXTPLACE := NEXTPLACE^ NEXT;
958 79 2 WRITELN(TURNS, 'DUMP');
959 96 2 IF DISPLAY = NOTWRITE THEN
960 102 2 WRITELN('DUMP');
961 119 2 LISTTURNS;
962 123 2 END;
963 125 1 ELSE NOMORE := TRUE;
964 127 1 END;(*DUMP*)
965 130 1 END;(*LEVEL6*)
966 7 0 0 PROCEDURE LEVEL6;
967 68 0 0
968 0 0
969 0 1 BEGIN
970 0 1 BEGIN
971 27 1 END;(*LEVEL6*)
972 56 0
973 0 0 PROCEDURE LEVEL5;
974 0 0
975 0 0 VAR
976 0 0
977 0 0 STOP5 := BOOLEAN;
978 0 0
979 0 1 BEGIN
980 0 1 STOP5 := FALSE;
981 3 1 STACKPOINTER := STACKPOINTER + 1;
982 9 1 CASE DIRECTION[STACKPOINTER - 2] OF
983 35 1 WEST : DIRECTION[STACKPOINTER] := EAST;
984 61 1 EAST : DIRECTION[STACKPOINTER] := WEST;
985 87 1 NORTH : DIRECTION[STACKPOINTER] := SOUTH;
986 113 1 SOUTH : DIRECTION[STACKPOINTER] := NORTH;
987 138 1 ELSE:
988 140 1 END;(*CASE*)
989 152 1 WHILE NOT STOP5 DO
990 156 2 BEGIN
991 156 2 BEGIN
992 182 2 IF NOT OBSTRUCT THEN UPDATE
993 193 2 ELSE LEVEL6;
I

1003
1007
1010
1014
1018
1021
1024
1027
1030
1033
1036
1039
1042
1045
1048
1051
1054
1058
1061
1064
1067
1070
1073
1076
1079
1082
1085
1088
1091
1094
1097
1099
1102
1105
1108
1111
1114
1117
1120
1123
1126
1129
1132
1135
1138
1141
1144
1147
1150
1153
1156
1159
1162
1165
1168
1171
1174
1177
1180
1183
1186
1189
1192
1195
1198
1201
1204
1207
1210
1213
1216
1219
1222
1225
1228
1231
1234
1237
1240
1243
1246
1249
1252
1255
1258
1261
1264
1267
1270
1273
1276
1279
1282
1285
1288
1291
1294
1297
1300
1303
1306
1309
1312
1315
1318
1321
1324
1327
1330
1333
1336
1339
1342
1345
1348
1351
1354
1357
1360
1363
1366
1369
1372
1375
1378
1381
1384
1387
1390
1393
1396
1399
1402
1405
1408
1411
1414
1417
1420
1423
1426
1429
1432
1435
1438
1441
1444
1447
1450
1453
1456
1459
1462
1465
1468
1471
1474
1477
1480
1483
1486
1489
1492
1495
1498
1501
1504
1507
1510
1513
1516
1519
1522
1525
1528
1531
1534
1537
1540
1543
1546
1549
1552
1555
1558
1561
1564
1567
1570
1573
1576
1579
1582
1585
1588
1591
1594
1597
1600
1603
1606
1609
1612
1615
1618
1621
1624
1627
1630
1633
1636
1639
1642
1645
1648
1651
1654
1657
1660
1663
1666
1669
1672
1675
1678
1681
1684
1687
1690
1693
1696
1699
1702
1705
1708
1711
1714
1717
1720
1723
1726
1729
1732
1735
1738
1741
1744
1747
1750
1753
1756
1759
1762
1765
1768
1771
1774
1777
1780
1783
1786
1789
1792
1795
1798
1801
1804
1807
1810
1813
1816
1819
1822
1825
1828
1831
1834
1837
1840
1843
1846
1849
1852
1855
1858
1861
1864
1867
1870
1873
1876
1879
1882
1885
1888
1891
1894
1897
1900
1903
1906
1909
1912
1915
1918
1921
1924
1927
1930
1933
1936
1939
1942
1945
1948
1951
1954
1957
1960
1963
1966
1969
1972
1975
1978
1981
1984
1987
1990
1993
1996
1999
2002
2005
2008
2011
2014
2017
2020
2023
2026
2029
2032
2035
2038
BEGIN

icient"

END;

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;

BEGIN

END;
FOR I := 1 TO L DO
  CASE TEMPPARRAY[I,J] OF
    'I', 'D', 'X' : TEMPPARRAY[I,J] := ' ';
    OTHERWISE : TEMPPARRAY[I,J] := 0;
  END;(*CASE*)

CASE TRIALNUMB OF
  XX : IF TEMPPARRAY[I,J] = 'X' THEN
    TEMPPARRAY[I,J] := ' ';  
  ONE : IF TEMPPARRAY[I,J] = '1' THEN
    TEMPPARRAY[I,J] := ' ';  
  TWO : IF TEMPPARRAY[I,J] = '2' THEN
    TEMPPARRAY[I,J] := ' ';  
  THREE : IF TEMPPARRAY[I,J] = '3' THEN
    TEMPPARRAY[I,J] := ' ';  
  FOUR : IF TEMPPARRAY[I,J] = '4' THEN
    TEMPPARRAY[I,J] := ' ';  
  FIVE : IF TEMPPARRAY[I,J] = '5' THEN
    TEMPPARRAY[I,J] := ' ';  
  SIX : IF TEMPPARRAY[I,J] = '6' THEN
    TEMPPARRAY[I,J] := ' ';  
  SEVEN : IF TEMPPARRAY[I,J] = '7' THEN
    TEMPPARRAY[I,J] := ' ';  
  END;(*CASE*)
```pascal
1441172
GETLM(L,M);
14421622
END;
14431841END;(*CASE*)
14441991END;(*WHICHDISPLAY*)
14452550PROCEDURE GETDISPLAY(VAR DISPLAY : OPTIONS);
144600PROCEDURE GETDISPLAY(VAR DISPLAY : OPTIONS);
1447322CASE TEMPCHAR OF
1448322 'V' : BEGIN
1449322 L := 88;
1450362 M := 30;
1451402DISPLAY := WRITETOSCREEN;
1452472END;
1453492READLN(ENVIRON,L);
1454632READLN(ENVIRON,M);
1455772DISPLAY := NOTWRITE;
1456842END;
1457862END;(*CASE*)
14581011END;(*GETDISPLAY*)
14591420PROCEDURE CONDITIONS;
146000PROCEDURE CONDITIONS;
1461000PROCEDURE CONDITIONS;
146201BEGIN
146301L := 88;
146451M := 30;
146591DUMPED := FALSE;
1466121WHITELM('ARE NEW OBJECTS REQUIRED?');
1467391REPEAT READLN(TEMPCHAR);
1468571UNTIL EOLN(INPUT);
1469811CASE TEMPCHAR OF
1470852'T' : BEGIN
1471852REWRITE(ENVIRON,'ENVIRON');
14721102WHICHDISPLAY(DISPLAY);
14731162INITIALISE;
14741202GETOBJECTS;
14751242GETDESTINATIONS;
14761282END;
14771302'C' : BEGIN
14781302RESET(ENVIRON,'ENVIRON');
14791552GETDISPLAY(DISPLAY);
14801622INITIALISE;
14811652GETENVIRON;
14821692COPY(CODEARRAY,ROBOTARRAY);
14831772END;
14841791OTHERWISE
14851792BEGIN
14861792RESET(ENVIRON,'ENVIRON');
14872042GETDISPLAY(DISPLAY);
14882102INITIALISE;
14892142GETENVIRON;
14902182END;
14912201END;(*CASE*)
14922481REWRITE(ENVIRON, 'ENVIRON');
14932721REWRITE(REFILE, 'REFILE');
14942971CLEAR(RESMATRIX);
14953031ORDCOPY(RESMATRIX);
14963091LASTPLACE := LISTHEAD;
14973211DESTINATION := LISTHEAD;
14983331NEXTPLACE := DESTINATION*.NEXT;
14993531STOPFINAL := FALSE;
15003561RESETPROG;
15013601END;(*CONDITIONS*)
15024370PROCEDURE BRANCH(DECISION : TYPEOFDECISION);
150300PROCEDURE BRANCH(DECISION : TYPEOFDECISION);
150400PROCEDURE BRANCH(DECISION : TYPEOFDECISION);
150501BEGIN
150601STORESTART(XCOORDIN,YCOORDIN);
1507171TRIALNMB := ONE;
1508201REPEAT
1509202BEGIN
1510202STOPBLOCK1 := FALSE;
1511232RESETSTART;
1512272METOBJECT := FALSE;
1513302WHILE NOT STOPBLOCK1 DO
1514352BLOCK1(CODEARRAY,DISPLAY,ENABLE,ROBOTARRAY,DECISION);
1515822END;
1516822RESULTS;
1517861TRIALNMB := SUCC(TRIALNMB);
15181071UNTIL NOT METOBJECT OR (TRIALNMB = SEVEN);
15191201END;(*BRANCH*)
15201600PROCEDURE ROUTEFOLLOW(DECISION : TYPEOFDECISION);
152200PROCEDURE ROUTEFOLLOW(DECISION : TYPEOFDECISION);
152301BEGIN
152401CONSULT(DISPLAY);
1525181RESETPROG;
1526221RESETSTART;
1527261STOPBLOCK1 := FALSE;
1528291DUMPED := FALSE;
1529321TRIALNMB := XX;
1530351WHILE NOT STOPBLOCK1 DO
1531401BLOCK1(CODEARRAY,DISPLAY,DISABLE,ROBOTARRAY,DECISION);
1532871CONSULT(DISPLAY);
15331041RESULTS;
15341081END;(*ROUTEFOLLOW*)
15351490PROCEDURE ROUTEFOLLOW(DECISION : TYPEOFDECISION);
153600BEGIN
1537001BEGIN
1538001NEW-BEGINNING;
1539181REPEAT
1540161BEGIN
```
ROUTEFOLLOW(PRIMARY);

IF DESTINATION^.NEXT = NIL THEN
  STOPFINAL := TRUE
ELSE
  COMPARE;
UNTIL STOPFINAL;
END.
APPENDIX B

THE QUASI-CONTINUOUS ENVIRONMENT PROGRAM LISTING

The extensions to the PASCAL language implemented by Hewlett-Packard [1981] enabled procedures to be compiled separately and then linked together for execution. To achieve this each procedure was stored in a separate file, complete with program header and data definitions. The procedure was then written and the file ended with a full stop, the main loop of a normal PASCAL program being omitted.

EXTERNAL procedures

If a procedure defined in this way used other procedures they were denoted as being EXTERNAL. The external procedure heading was defined as normal and was then succeeded by the EXTERNAL reserved word. This notified the compiler that the previously defined procedure was in a different file, which would be available when linking.

EXTVARiables

If variables were defined in one PASCAL program and used in others they were specified by compiler options. If the variable was defined in a different file then it was preceded by $EXTVAR ON$ and succeeded by $EXTVAR OFF$. Similarly, if a variable was to be used in other PASCAL programs it was preceded by $GLOBVAR ON$ and succeeded by $GLOBVAR OFF$ to identify it as a global variable.
The compiler options described above were defined within the body of the PASCAL programs themselves. In addition, there were eight compiler options given at the beginning of each file which were specific to the microprocessor being used.

"8086"

The compiler generated relocatable object code specific for a microprocessor and the actual microprocessor was specified on the first line of the program. For this particular project the 8086 16 bit microprocessor was used.

$POINTER SIZE 32$

When accessing memory the 8086 could use 16 or 32 bit memory pointers. The 32 bit pointers were used to allow access to the complete 1MByte memory space.

$FAR LIBRARIES ON$

The PASCAL language included standard arithmetic, trigonometric and logical functions (i.e. sine, cosine, etc) which were available as assembled relocatable object files, kept in library files. Two types of library file were available, those which only saved a 16 bit return address on the stack when subroutines were called, and those which saved a 32 bit return address (i.e. FAR_LIBRARIES). FAR_LIBRARIES were used as they could be accessed anywhere within the 1MByte address space.

$FAR_PROC ON$

To ensure that PASCAL procedures were called which saved a 32 bit return address the FAR_PROC compiler option was used, to enable the procedures to be called from anywhere in the 1MByte address space.

$EXTENSIONS ON$
This compiler option allowed the extensions to the PASCAL language defined initially (i.e. EXTERNAL, EXTVAR, etc) to be used.

$DEBUG$ ON$

When initially writing a PASCAL program it was usually necessary to perform checks on the flow of the program, to prevent situations which had not been envisaged causing an irrecoverable error (e.g. an undefined item in a case statement) or an undetectable error. With the DEBUG option ON extra code was added to high risk statements, such as the CASE statement, which detected such errors. The penalty for this was the additional memory space required and the longer execution time as there was more code to execute. Once the program was tested the DEBUG option could be turned OFF to reduce these penalties.

$GLOBPROC$ ON$

To enable a procedure defined in one file to be used in another, it was defined as a global procedure by using this option.

$SEPARATE CONST$ OFF$

The compilation of PASCAL programs generated numeric constants (i.e. -1, pi, etc) which needed to be included in the program code area of memory to obtain a permanent copy of them. The 8086 normally assumes that all data, including constants, is kept in a different area of memory from the program code. The separation is automatically performed by the compiler, putting all the variables (i.e. those which use Read/Write Memory) and constants (i.e. those which use Read Only Memory) into the same memory area, so that constant values will be lost when the power is turned off. This was avoided by specifying that the compiler option
SEPARET_CONST should be OFF so that the constants remained in the program code area, and would then be permanently saved in ROM.

The program structure

The flow of the program and the reasoning behind the method of design are described in detail in chapters two and four. The complete program listing is given on pages B6 to B32 of this appendix.

The structure of the program is shown in Figure B.1 in a top-down diagram using procedure names to identify the processes involved. The procedures were called or executed by the procedure connected immediately above them, so that MOVE_ROB and ONE_PULSE were called from the procedure SCAN_ENVIRON. This allowed the position of every significant procedure within the program flow to be identified.

The top level of the program was executed by PROTO and split into four groups; path finding, position location, initialisation and utility procedures.

The path finding process can be traced via FIND_PATH and the position location via FIND_POSITION. The initialisation procedures were INITI_8251 (which initialised the USART), INITIALISE, SET_UP_WORLD, GO_TO, MOVE_GET and the utility procedures were PRINT_SCAN, LIST_ARRAY (which output the range data to the VDU), and SCAN_ENVIRON and ONE_PULSE (which were the procedures used to obtain environment data from the simulated or prototype mobile robots).
Figure B.1 The structure of the program PROTO
- Identifies procedures which appear more than once
- Identifies assembly language subroutines
$8086$

```
$POINTER_SIZE 32$
$FAR_LIBRARIES ON$
$FAR_PROC ON$
$EXTENSIONS ON$
$DEBUG ON$
$SEPARATE_CONST OFF$

PROGRAM PROTO;

TYPE
VECTOR = RECORD XCOOR,YCOOR,ANGLE : REAL END;
SCAN_ARRAY = ARRAY[0..63] OF REAL;
INTP = INTEGER;(*USED TO INIT HEAP*)

CONST
HEAPSIZE = UNSIGNED_16(0EFFH);(*DEFINES THE HEAP SIZE WHICH IN CONTAINED IN MOB8610A*)
PI = 3.14159;
TWO_PI = 6.28318;
HALF_PI = 1.57080;
ZERO = 0.0;
VAR
TEMP_RANGE : REAL;

$GLOBVAR ON$
ROBOT : VECTOR;
SENSOR : VECTOR;
SIMULATION : BOOLEAN;

$GLOBVAR OFF$
COMMAND : CHAR;
END_ALL : BOOLEAN;

$EXTVAR ON$
HEAPSTART : INTEGER;(*DEFINES BEGINING OF HEAP:IN MOB8610A*)

PROCEDURE WRITE_CHAR(S : STRING); EXTERNAL;
PROCEDURE WRITE_LINE(S : STRING); EXTERNAL;
PROCEDURE SET_UP_WORLD; EXTERNAL;
PROCEDURE PRINT_SCAN; EXTERNAL;
PROCEDURE LIST_ARRAY; EXTERNAL;
PROCEDURE FIND_PATH; EXTERNAL;
PROCEDURE MOVE_GET(RATIO : INTEGER); EXTERNAL;
PROCEDURE FIND_POSITION; EXTERNAL;

PROCEDURE INITIALIZE;
BEGIN(*INITIALISE THE POSITION OF THE MOBILE ROBOT*)
WITH ROBOT DO
BEGIN
XCOOR := 500.0;
YCOOR := 500.0;
ANGLE := HALF_PI;
END;(*WITH*)
SENSOR := ROBOT;
END;(*INITIALISE*)

PROCEDURE GO_TO(VAR ROBOT: VECTOR);
BEGIN(*ALTERS THE POSITION OF THE MOBILE ROBOT*)
WRITE CHAR('XCOOR ');
READLN REAL(ROBOT.XCOOR);
WRITE CHAR('YCOOR ');
READLN REAL(ROBOT.YCOOR);
ROBOT.ANGLE := 0.0;
END;

PROCEDURE GO_TO(VAR ROBOT: VECTOR);

PROCEDURE INIT_8251; EXTERNAL;(*INITIALISE THE USART*)

PROCEDURE INITIALIZE;
BEGIN
INC 8251;(*INITIALISE THE USART*)
INITIALIZE;
TEMP_RANGE := 0.0;
END_ALL := FALSE;
SIMULATION := FALSE;
(*OBTAIN THE PROCESS REQUIRED FROM THE USER*)
REPEAT(*UNTIL END*)
BEGIN
WRITE CHAR('ENTER COMMAND ');
READLN CHAR(COMMAND);
CASE COMMAND OF
'S' : SCAN_ENVIRON;
'T' : IF SIMULATION = FALSE THEN SIMULATION := TRUE ELSE SIMULATION := FALSE;
'M' : MOVE_GET(1);
'W' : SET_UP_WORLD;
'G' : GO_TO(ROBOT);
'F' : FIND_PATH;
'P' : FIND_POSITION;
END;
END;
```

```
'I' : INITIALISE;
OTHERWISE BEGIN WRITE_CHAR(COMMAND); WRITELN_CHAR('UNKNOWN
COMMAND');
END;
END;(*CASE*)
UNTIL END_ALL;
END.(*END OF THE PROGRAM*)

"8086"
$POINTER_SIZE 32G
$FAR_LIBRARY ON
$FAR_PROC ON
$DEBUG ON
$SEPARATE_CONST OFF

PROGRAM RANGE_TO_OBJECT;

TYPE
LINE = ARRAY[0..64,0..3] OF REAL;
VECTOR = RECORD XCOORD, YCOORD, ANGLE : REAL END;

VAR

SENSOR : VECTOR;

PROCEDURE CALC_INTERSECT(VAR RANGE : REAL;
X1,Y1,X2,Y2 : REAL); EXTERNAL;

PROCEDURE RANGE_TO_OBJECT(VAR WORLD : LINE; VAR NUMB_LINES : INTEGER;
VAR OBJ_RANGE : REAL);

CONST
X1_POINT = 0;
Y1_POINT = 1;
X2_POINT = 2;
Y2_POINT = 3;
MAX_RANGE = 500.0; (*500 CM*)

VAR

TEST_RANGE : REAL;
NEXT_LINE : INTEGER;

BEGIN
OBJ_RANGE := MAX_RANGE + 1.0;
FOR NEXT_LINE := 1 TO NUMB_LINES DO
BEGIN(*FOR EVERY OBJECT SIDE CALCULATE THE INTERSECT POINT*)
WITH SENSOR DO
CALC_INTERSECT(TEST_RANGE,WORLD[NEXT_LINE,X1_POINT]-XCOORD,
WORLD[NEXT_LINE,Y1_POINT]-YCOORD,
WORLD[NEXT_LINE,X2_POINT]-XCOORD,
WORLD[NEXT_LINE,Y2_POINT]-YCOORD);
IF OBJ_RANGE > TEST_RANGE THEN OBJ_RANGE := TEST_RANGE;
END;(*FOR*)
END;(*RANGE_TO_OBJECT*)
PROGRAM CALC_INTERSECT;

TYPE

VECTOR = RECORD XCOOR,YCOOR,ANGLE : REAL END;

CONST MAXRANGE = 500.0; (*500 CM*)

VAR

XDIFF,NOTZERO : BOOLEAN;

WITHIN SIDE : BOOLEAN;

M_OBJECT,C_OBJECT,M_SENSOR : REAL;

XINTERSECT,YINTERSECT : REAL;

$EXTVAR ON$

SENSOR : VECTOR;

$EXTVAR OFF$

PROCEDURE CHECK_INTERSECT(INTERSECT,FIRST_POINT,SECOND_POINT : REAL;

VAR WITHIN SIDE : BOOLEAN);

EXTERNAL;

PROCEDURE INFRONT_CHECK(XINTERSECT,YINTERSECT : REAL;

VAR WITHIN SIDE : BOOLEAN); EXTERNAL;

PROCEDURE CALC_INTERSECT(VAR RANGE : REAL;

X1,Y1,X2,Y2 : REAL);

BEGIN(*CHECKS TO PREVENT DIVIDE BY ZEROS*)

WITHIN SIDE := TRUE; (*INITIALISE*)

IF ABS(X1-X2) < 0.01 THEN XDIFF := TRUE ELSE XDIFF := FALSE;

IF ABS(COS(SENSOR.ANGLE)) > 0.01 THEN NOTZERO := TRUE ELSE NOTZERO := FALSE;

IF XDIFF AND NOTZERO THEN (*FIND STRAIGHT LINE EQUATIONS OF LINE SEGMENT AND SENSOR BEAM*)

BEGIN

M_OBJECT := (Y1-Y2)/(X1-X2);

C_OBJECT := Y1 - (M_OBJECT * X1);

M_SENSOR := (SENSOR.ANGLE) / COS(SENSOR.ANGLE);

(*NOW SOLVE THE SIMULTANEOUS EQUATIONS*)

BEGIN

XINTERSECT := C_OBJECT/(M_SENSOR - M_OBJECT);

YINTERSECT := M_SENSOR * XINTERSECT;

END;

ELSE IF INFINITE LINE SEGMENT GRADIENT USE A DIFFERENT METHOD*:

IF NOT XDIFF AND NOTZERO THEN

BEGIN

XINTERSECT := X1;

YINTERSECT := SIN(SENSOR.ANGLE) * XINTERSECT/COS(SENSOR.ANGLE);

END;

ELSE (*IF A ZERO SENSOR GRADIENT*)

IF XDIFF AND NOTZERO THEN

BEGIN

XINTERSECT := 0.0;

M_OBJECT := (Y1-Y2)/(X1-X2);

C_OBJECT := Y1 - (M_OBJECT * X1);

IF ABS(M_OBJECT) < 0.01 THEN YINTERSECT := Y1

ELSE YINTERSECT := (XINTERSECT - C_OBJECT)/M_OBJECT;

END

ELSE (*IF THIS LINE IS EXECUTED THE TWO LINES ARE EFFECTIVELY PARALLEL*)

WITHIN SIDE := FALSE;

(*NOW PERFORM CHECKS THAT INTERSECTION IS WITHIN THE OBJECT SIDE AND INFRONT OF THE SENSOR. THE X AND Y CHECKS ARE PERFORMED SEPARATELY*)

CHECK_INTERSECT(XINTERSECT,X1,X2,WITHIN SIDE);

CHECK_INTERSECT(YINTERSECT,Y1,Y2,WITHIN SIDE);

(*CHECK INFRONT OF SENSOR*)

INFRONT_CHECK(XINTERSECT,YINTERSECT,WITHIN SIDE);

IF WITHIN SIDE THEN (*CALCULATE THE RANGE TO INTERSECTION*)

RANGE := SQRT((XINTERSECT - XINTERSECT)^2 + (YINTERSECT - YINTERSECT)^2)

ELSE RANGE := MAXRANGE + 1.0;

END;(*CALC_INTERSECT*)
PROGRAM CHECK_INTP;

PROCEDURE CHECK_INTERSECT(INTERSECT,FIRST_POINT,SECOND_POINT : REAL;
    VAR WITHIN_SIDE : BOOLEAN);

BEGIN
    IF WITHIN_SIDE THEN
        IF (ABS(FIRST_POINT - SECOND_POINT) < 1.00) THEN (*CHECK IF THE TWO POINTS ARE CLOSE TOGETHER*)
            BEGIN
                IF (ABS(INTERSECT - FIRST_POINT) > 1.00) THEN (*IF THEY ARE DO ONLY A SIMPLE INTERSECTION TEST*)
                    WITHIN_SIDE := FALSE;
                END;
            END;
    ELSE (*IF NOT CLOSE TOGETHER PERFORM THE DETAILED CHECK*)
        IF (((INTERSECT > FIRST_POINT) AND (INTERSECT > SECOND_POINT)) OR ((INTERSECT < FIRST_POINT) AND (INTERSECT < SECOND_POINT))) THEN
            WITHIN_SIDE := FALSE;
    END;(*CHECK_INTERSECT*)

END;

PROGRAM INFRONT_CHECKP;

CONST
    PI = 3.14159;
    TWOPI = 6.28318;
    HALF_PI = 1.57080;
    ONE_AN_HALF_PI = 4.71239;

TYPE
    VECTOR = RECORD
        XCOORD,YCOORD,ANGLE : REAL
    END;

PROCEDURE INFRONT_CHECK(XINTERSECT,YINTERSECT : REAL;
    VAR WITHIN_SIDE : BOOLEAN);

    VAR ALPHA : REAL; (*TEMP STORE FOR ROBOT.ANGLE TO PREVENT IT BEING CHANGED*)

BEGIN
    IF WITHIN_SIDE THEN (*IF STILL VALID INTERSECT THEN CARRY ON*)
        BEGIN
            ALPHA := SENSOR.ANGLE;
            IF ALPHA < 0.0 THEN ALPHA := ALPHA + TWOPI;
            ELSE IF ALPHA > TWOPI THEN ALPHA := ALPHA - TWOPI;
            (*THE FOLLOWING CHECKS THAT THE INTERSECT POINT IS IN FRONT OF THE SENSOR*)
            BY USING THE SENSOR ANGLE TO DETERMINE WHAT THE SIGNS OF THE INTERSECT X,Y VALUES SHOULD BE (DEPENDS ON PARTICULAR QUADRANT) AND THEN CHECKS THAT THEY ARE ACCEPTABLE*)
            IF (0.0 < ALPHA) AND (ALPHA < HALF_PI) THEN
                (*FIRST QUADRANT, BOTH X,Y SHOULD BE POSITIVE*)
    END;(*INFRONT_CHECK*)
BEGIN
   IF (XINTERSECT < 0.0) OR (YINTERSECT < 0.0) THEN
      WITHIN_SIDE := FALSE;
   END
   ELSE (*IF SECOND QUADRANT x SHOULD BE NEG AND y POS*)
      IF (HALF_PI < ALPHA) AND (ALPHA < PI) THEN
         IF (XINTERSECT > 0.0) OR (YINTERSECT < 0.0) THEN
            WITHIN_SIDE := FALSE;
         END
      END (*IF THIRD QUADRANT BOTH x,y ARE NEG*)
      IF (PI < ALPHA) AND (ALPHA < ONE_AN_HALF_PI) THEN
         IF (XINTERSECT < 0.0) OR (YINTERSECT > 0.0) THEN
            WITHIN_SIDE := FALSE;
         END (*ASSUMES THAT They ARE IN FOURTH QUADRANT*)
      END (*IF WITHIN SIDE*)
   END (*INFRONT_CHECK*)

"8086"
SPINTER SIZE 32S
$FAR_LIBRARIES ON
$FAR_PROC ON
$EXTENSIONS ON$
$DEBUG ON
$GLOBPROC ON
$SEPARATE_CONST OFF$

PROGRAM PONE_PULSE;

TYPE
   VECTOR = RECORD XCOORD, YCOORD, ANGLE : REAL END;
   BUFFER = ARRAY[1..16] OF BYTE;
   LINE_ARRAY = ARRAY[1..64,0..3] OF REAL;

   CONST
   ROFFSET = 43.0;

   VAR
   DSTACK : BUFFER;
   OVER_BITE,BITE : BYTE;
   ADDRESS,NUMBYTES : BYTE;
   SIMULATION : BOOLEAN;
   SENSOR,ROBOT : VECTOR;
   WORLD : LINE_ARRAY;
   NUMBLINES : INTEGER;

   SEND_MODULE : INTEGER;

PROCEDURE WRITELN_REAL(NMBR : REAL); EXTERNAL;
PROCEDURE SEND; EXTERNAL;
PROCEDURE RECEIVE; EXTERNAL;
PROCEDURE CON_SIGNED_8; EXTERNAL;
PROCEDURE RANGE_TO_OBJECT(VAR WORLD : LINE_ARRAY;
   VAR NUMBLINES : INTEGER;
   VAR OBJ_RANGE : REAL); EXTERNAL;
PROCEDURE ONE_PULSE(VAR RANGE : REAL);

BEGIN
   IF SIMULATION THEN(*FOR THE SIMULATED RANGEFINDER USE *)
      BEGIN (*THE SIMULATED ENVIRONMENT*)
SENSOR := ROBOT;
RANGE_TO_OBJECT(WORLD, NUMBLINES, RANGE);
END
ELSE
BEGIN
(*INITIATE PROTOTYPE RANGEFINDER*)
ADDRESS := 4;
NUMBYTES := 1;
DSTACK[1] := 1;
SEND;
RECEIVE; (*WAIT FOR RANGE*)
SEND MODULE := DSTACK[1];
(*SAVE RANGE IN ARRAY*)
IF SEND MODULE = 4 THEN
BEGIN (*CONVERT BYTE TO REAL*)
BITE := DSTACK[4];
CON SIGNED 8; (*ASSM ROUTINE TO CONVERT TO SIGNED_8*)
RANGE := (128.0 * OVER_BITE) + BITE;
(*CONVERT RANGE TO CM*)
RANGE := RANGE * 2/3;
RANGE := RANGE + OFFSET;
END; (*IF*)
END;
WRITELN REAL(RANGE);
END;
(*ONE_PULSE*)

"8086"
$POINTER_SIZE 32$
$FAR_LIBRARIES ON$
$FAR_PROC ON$
$EXTENSIONS ON$
$DEBUG ON$
$SEPARATE_CONST OFF$

PROGRAM PSCAN_ENVIRON;

TYPE
SCAN ARRAY = ARRAY(0 •• 63) OF REAL;

CONST
PI = 3.14159;
WHEELBASE = 17.20;

VAR
SCAN_ARRAY = ARRAY[0 .. 63] OF REAL;

PROCEDURE SCAN_ENVIRON;
BEGIN
(*TURN 90 DEGREES TO RIGHT*)
MOVEMENT := 'R';
DIST_REAL := PI/2 * WHEELBASE;
MOVE_ROB(MOVEMENT, DIST_REAL, 1);
(*SLOW SCAN LEFT FOR 180 DEGREES*)
DIST_REAL := 1.688;
MOVEMENT := 'L';
FOR CUM DIST := 0 TO 31 DO
BEGIN
MOVE_ROB(MOVEMENT, DIST_REAL, 1);
ONE_PULSE(RANGE_REAL);
END;
**CD.... ....
\[ \text{RANGE\_ARRAY}[\text{CUM\_DIST}] := \text{RANGE\_REAL}; \]
\[ \text{WRITE\_REAL}(\text{RANGE\_REAL}); \]
\[ \text{END}; \]

(*RETURN TO ORIGINAL HEADING*)

\[ \text{MOVEMENT} := \text{'}R\'; \]
\[ \text{DIST\_REAL} := \pi/2 \times \text{WHEELBASE}; \]
\[ \text{MOVE\_ROB}(\text{MOVEMENT}, \text{DIST\_REAL}, 1); \]
\[ \text{END}; \text{(*SCAN\_ENVIRON*)} \]

"8986"

\$\text{POINTER\_SIZE 128}\$
\$\text{FAR\_LIBRARIES\_ONS}\$
\$\text{FAR\_PROC\_ONS}\$
\$\text{EXTENSIONS\_ONS}\$
\$\text{DEBUG\_ONS}\$
\$\text{GLOB\_PROC\_ONS}\$
\$\text{SEPARATE\_CONST\_OFFS}\$

\[ \text{PROGRAM\ LIST\_ARRAYP}; \]

\[ \text{TYPE} \]
\[ \text{SCAN\_ARRAY} = \text{ARRAY}[0..63] \text{OF} \text{REAL}; \]

\[ \text{VAR} \]
\[ \text{RANGE\_ARRAY} : \text{SCAN\_ARRAY}; \]
\[ \text{DIFF\_ARRAY} : \text{SCAN\_ARRAY}; \]

\[ \text{PROCEDURE\ WRITE\_CHAR(S : STRING); EXTERNAL;} \]
\[ \text{PROCEDURE\ WRITE\_LN\_CHAR(S : STRING); EXTERNAL;} \]
\[ \text{PROCEDURE\ WRITE\_INT(NMBR : INTEGER); EXTERNAL;} \]
\[ \text{PROCEDURE\ WRITE\_REAL(NMBR : REAL); EXTERNAL;} \]
\[ \text{PROCEDURE\ READLN\_CHAR(VAR CH : CHAR); EXTERNAL;} \]

\[ \text{PROCEDURE\ LIST\_ARRAY}; \]

\[ \text{VAR} \]
\[ \text{WHICH\_ARRAY} : \text{CHAR}; \]
\[ \text{COUNT, INDEX} : \text{INTEGER}; \]
\[ \text{PRINT\_ARRAY} : \text{SCAN\_ARRAY}; \]

\[ \text{BEGIN} \text{(*OUTPUT THE SELECTED ARRAY RANGE VALUES TO THE VDU*)} \]
\[ \text{WRITE\_CHAR('ARRAY ')}; \]
\[ \text{READLN\_CHAR(WHICH\_ARRAY)}; \]
\[ \text{CASE WHICH\_ARRAY \text{OF}} \]
\[ \text{'R' : PRINT\_ARRAY} \text{:= RANGE\_ARRAY}; \]
\[ \text{'D' : PRINT\_ARRAY} \text{:= DIFF\_ARRAY}; \]
\[ \text{OTHERWISE PRINT\_ARRAY} \text{:= RANGE\_ARRAY}; \]
\[ \text{END}; \text{(*CASE*)} \]
\[ \text{INDEX} := 0; \]
\[ \text{REPEAT} \text{(*UNTIL INDEX} \text{> 63*)} \]
\[ \text{WRITE\_INT(INDEX)}; \]
\[ \text{COUNT} := 0; \]
\[ \text{REPEAT} \text{(*UNTIL 'END-OF-LINE'*)} \]
\[ \text{WRITE\_REAL(PRINT\_ARRAY(INDEX))}; \]
INDEX := INDEX +1;
COUNT := COUNT +1;
UNTIL (COUNT > 9) OR (INDEX > 63);
WRITELN_CHAR(' ');
UNTIL INDEX > 63;
WRITELN_CHAR('FINISHED LISTING');
END; (*LIST_ARRAY*)

"8086"
$POINTER_SIZE 32$
$FAR_LIBRARIES ON$
$FAR_PROC ON$
$EXTENSIONS ON$
$DEBUG ON$
$GLOB_PROC ON$
$SEPARATE_CONST OFF$

PROGRAM PRINT_SCANP;

TYPE
SCAN_ARRAY = ARRAY[0..63] OF REAL;

VAR
$EXTVAR ON$
RANGE_ARRAY : SCAN_ARRAY;

$EXTVAR OFF$

PROCEDURE WRITE_CHAR(S : STRING); EXTERNAL;
PROCEDURE WRITELN_CHAR(S : STRING); EXTERNAL;
PROCEDURE READLN_CHAR(VAR CH : CHAR); EXTERNAL;
PROCEDURE READLN_REAL(VAR NMBR : REAL); EXTERNAL;
PROCEDURE PRINT_SCAN;

VAR
VDU_SCALE, VDU_WIDTH, RANGE, TEMP_REAL : REAL;
VDU_ANS : CHAR;
COUNT, INDEX, START, FINISH : INTEGER;

BEGIN (*EACH RANGE VALUE O/P TO THE VDU AS A LINE OF SPACES
PROPORTIONAL TO THE RANGE. EACH LINE IS TERMINATED BY
A * CHARACTER*)
WRITE_CHAR('CHANGE SCALE? ');
READLN_CHAR(VDU_ANS);
IF VDU_ANS = 'Y' THEN
BEGIN (*IS THE SCALE FACTOR TO BE CHANGED?*)
WRITE_CHAR('SCALE? ');
READLN_REAL(VDU_SCALE);
WRITE_CHAR('VDU-WIDTH? ');
READLN_REAL(VDU_WIDTH);
VDU_WIDTH := VDU_WIDTH - 1.0;
END
ELSE
BEGIN (*STANDARD SCALE VALUES*)
VDU_ANS := 'Y';
VDU_SCALE := 7.0;
VDU_WIDTH := 79.8;
END;/*IF*/
WHILE VDU_ANS = 'Y' DO
BEGIN
WRITE_CHAR('START? ');
READLN_REAL(TMP_REAL);
START := ROUND(TMP_REAL);
WRITE_CHAR('FINISH? ');
READLN_REAL(TMP_REAL);
FINISH := ROUND(TMP_REAL);
FOR INDEX := START TO FINISH DO
BEGIN
RANGE := (RANGE ARRAY[index]/VDU_SCALE) - 0.5;/*AVoids rounding up over run*/
IF RANGE < VDU_WIDTH THEN(*OUTPUT THE SPACES*)
FOR COUNT := Ø TO ROUND(RANGE) DO WRITE_CHAR(' ')
ELSE FOR COUNT := 1 TO ROUND(VDU_WIDTH) DO WRITE_CHAR(' ');
WRITELN_CHAR('**');
END;
WRITE_CHAR('MORE? ');
REWriteln CHAR(VDU_ANS);
END;/*WHILE*/
WRITELN_CHAR('FINISHED PRINTING');
END;/*PRINT_SCAN*/

"8886"
$POINTER SIZE 32$
$FAR_LIBRARIES ON$
$FAR_PROC ON$
$EXTENSIONS ON$
$DEBUG ON$
$GLOBPROC ON$
$SEPARATE_CONST OFF$
PROGRAM MOVE_ROBP;

TYPE
VECTOR = RECORD XCOORD,YCOORD,ANGLE : REAL END;
CONST
PI = 3.14159;
TWO_PI = 6.28318;
ZERO = 0.0;
WHEELBASE = 17.20;
DCONST1 = 6.41;
DCONST2 = 8.59;
DCONST3 = 9.81;
DCONST4 = 10.26;
DCONST5 = 10.77;
DCONST6 = 11.15;
DCONST7 = 11.35;
VAR
DIST_COUNT,DISTANCE : REAL;
REAL_STEPS : REAL;
TEMP_INT : INTEGER;
SEXTVAR ON$;
DSTACK : BUFFER;
ADDRESS,NUMBYTES : BYTE;
SIMULATION : BOOLEAN;
ROBOT : VECTOR;
SEXTVAR OFFS$;
PROCEDURE SEND; EXTERNAL;
PROCEDURE RECEIVE; EXTERNAL;
PROCEDURE WRITELN_CHAR(S : STRING); EXTERNAL;
PROCEDURE WRITE_REAL(NMBR : REAL); EXTERNAL;
PROCEDURE WRITE_REAL(NMBR : REAL); EXTERNAL;
PROCEDURE MOVF_ROB(MOVEMENT : CHAR;
REAL_DIST : REAL;
  RATIO : INTEGER);

BEGIN
  (*UPDATE SIMULATION FIRST*)
CASE MOVEMENT OF
  'F' : WITH ROBOT DO
    BEGIN
      XCOORD := XCOORD + (REAL_DIST*COS(ANGLE));
      YCOORD := YCOORD + (REAL_DIST*SIN(ANGLE));
    END; (*WITH*)
  'B' WITH ROBOT DO
    BEGIN
      XCOORD := XCOORD -(REAL_DIST*COS(ANGLE));
      YCOORD := YCOORD -(REAL_DIST*SIN(ANGLE));
    END; (*WITH*)
  'L' : WITH ROBOT DO
    ANGLE := ANGLE + (REAL_DIST/WHEELBASE);
  'R' WITH ROBOT DO
    ANGLE := ANGLE - (REAL_DIST/WHEELBASE);
END; (*CASE*)
  IF ROBOT.ANGLE > TWO_PI THEN
    WITH ROBOT DO
      REPEAT ANGLE := ANGLE - TWO_PI
      UNTIL ANGLE < TWO_PI
  ELSE
    IF ROBOT.ANGLE < ZERO THEN
      WITH ROBOT DO
        REPEAT ANGLE := ANGLE + TWO_PI
        UNTIL ANGLE > ZERO;
    WRITE_REAL(ROBOT.XCOORD); WRITE_REAL(ROBOT.YCOORD); WRITELN_REAL(ROBOT.ANGLE * 1000.0);

  IF NOT SIMULATION THEN
    BEGIN(*MOVE THE PROTOTYPE MOBILE ROBOT IF NOT SIMULATED*)
      (*CONVERT DISTANCE TO STEPS*)
      IF RATIO = 1 THEN REAL_STEPS := REAL_DIST * DCONST1
      ELSE IF RATIO = 2 THEN REAL_STEPS := REAL_DIST * DCONST2
      ELSE IF RATIO = 3 THEN REAL_STEPS := REAL_DIST * DCONST3
      ELSE IF RATIO = 4 THEN REAL_STEPS := REAL_DIST * DCONST4
      ELSE IF RATIO = 5 THEN REAL_STEPS := REAL_DIST * DCONST5
      ELSE IF RATIO = 6 THEN REAL_STEPS := REAL_DIST *
      DCONST6
      ELSE WRITELN_CHAR('ERR IN MOVE ROB');
      DISTANCE := ROUND(REAL_STEPS);
      DIST_COUNT := 255;
      WHILE DISTANCE > 255 DO
        BEGIN
          DIST_COUNT := DIST_COUNT - 1;
          DISTANCE := DISTANCE - 255;
        END;
  DISTANCE := 255 - DISTANCE;
  TEMP_INT := 96 + RATIO;
CASE MOVEMENT OF
  'F' : TEMP_INT := TEMP_INT + 16;
  'B' : TEMP_INT := TEMP_INT + 16 + 8;
  'L' : TEMP_INT := TEMP_INT + 8;
  'R' : TEMP_INT := TEMP_INT + 0;
  OTHERWISE TEMP_INT := TEMP_INT + 16;
END; (*CASE*)
  (*SEND TO MOTOR DRIVE CONTROLLER*)
  ADDRESS := 1;
  NUMBYTES := 3;
  DSTACK[1] := TEMP_INT;
  DSTACK[2] := DIST_COUNT;
  DSTACK[3] := DISTANCE;
  SEND; (*TO ROBOT*)
  (*WAIT FOR CONFIRMATION OF MOVEMENT*)
  RECEIVE; (*CONFIRMS MOVE COMPLETED*)
END; (*MOVE_ROB*)
PROGRAM AVOIDP;

VAR
  PRF_V DEST POS_PTR;

PROCEDURE AVOID(VAR FIRST_POS: POS_PTR;
                 VAR PATH_CLEAR: BOOLEAN);

VAR
  PREV_DEST: POS_PTR;

VECTOR = RECORD XCOOR,YCOOR,ANGLE: REAL END;

PLACE = RECORD
  XPOS,YPOS: REAL;
  NEXT_POS,PREV_POS: PLACE END;

CONST
  PI = 3.14159;
  TWO_PI = 6.2832;
  MINUS_PI = -3.14159;
  ZERO = 0.0;
  WHEEL_BASE = 17.2;

PROCEDURE MOVE_ROB(MOVEMENT: CHAR;
                    DISTANCE: REAL;
                    RATIO: INTEGER) ;EXTERNAL;

PROCEDURE ONE_PULSE(VAR RANGE: REAL); EXTERNAL;

FUNCTION SQR(NMBR: REAL) : REAL; EXTERNAL;

PROCEDURE WRITELN(CHAR(S: STRING)) ;EXTERNAL;

PROCEDURE WRITE_REAL(NMBR: REAL) ;EXTERNAL;

PROCEDURE WRITELN_REAL(NMBR: REAL) ;EXTERNAL;

PROCEDURE GET_INTER_DEST(VAR PREV_DEST: PLACE;
                          VAR PATH_CLEAR: BOOLEAN) ;EXTERNAL;

BEGIN
  ERR_THRES := 0.06;(*3.3 DEGREES*)
  PREV_DEST := FIRST_POS;
  REPEAT(*UNTIL AT FINAL DESTINATION*)
    REPEAT(*UNTIL AT INTER DEST*) (*ORIENTATE WITH NEXT DEST*)
      WITH PREV_DEST.NEXT_POS DO
        BEGIN
          XDIFF := XPOS - ROBOT.XCOOR;
          YDIFF := YPOS - ROBOT.YCOOR;
        END; (*WITH*)
          THETA := ARCTAN(YDIFF/XDIFF);
          (*SELECT CORRECT QUADRANT*)
          IF XDIFF < ZERO THEN THETA := THETA + PI
          ELSE IF (XDIFF > ZERO) AND (YDIFF < ZERO) THEN
            THETA := THETA + TWO_PI;
          THETA := THETA - ROBOT.ANGLE;
          IF THETA > PI THEN THETA := THETA - TWO_PI
          ELSE IF THETA < MINUS_PI THEN THETA := THETA + TWO_PI;
          IF ABS(THETA) > ERR_THRES THEN
            BEGIN
              IF THETA > ZERO THEN
                MOVEMENT := 'L'
              ELSE MOVEMENT := 'R';
              DISTANCE := WHEEL_BASE * ABS(THETA);
              MOVE_ROB(MOVEMENT,DISTANCE,1);
              ONE_PULSE(TEMP RANGE);
            END
          ELSE
            IF DISTANCE > RANGE_TO_DEST THEN
              BEGIN(*IF MORE THAN THRESHOLD DISTANCE MOVE FORWARD*)
                DISTANCE := RANGE_TO_DEST;
                MOVEMENT := 'F';
                MOVE_ROB(MOVEMENT,DISTANCE,1);
                ONE_PULSE(TEMP RANGE);
                WITH PREV_DEST.NEXT_POS DO
                  RANGE_TO_DEST := SQRT(SQR(ROBOT.XCOOR-XPOS)+SQR(ROBOT.YCOOR-YPOS));
              END
            ELSE BEGIN(*MOVE CLOSER IF NECESSARY*)
              DISTANCE := TEMP RANGE/2.0;
              IF DISTANCE > 55.0 THEN MOVE_ROB('F',DISTANCE,1);(*STOPS IT GETTING TO CLOSE*)
            END
          END
        END
      END
    END
  END
END
"8086"

$POINTER SIZE 32$
$FAR_LIBRARIES ON$
$FAR_PROC ON$
$EXTENSIONS ON$
$DEBUG ON$
$SGLOBPROC ON$
$SEPARATE_CONST OFF$

PROGRAM PGET_INTER_DEST;

TYPE
VECTOR = RECORD XCOOR,YCOOR,ANGLE : REAL END;
POS_PTR = PLACE;
PLACE = RECORD
XPOS,YPOS : REAL;
NEXT_POS,PREV_POS : POS_PTR;
END;

CONST
ZERO = 0.0;
HALF_WHEELBASE = 17.20;

VAR
$EXTVAR ON$
ROBOT : VECTOR;
$EXTVAR OFF$

PROCEDURE MOVE_ROB(MOVEMENT : CHAR; DISTANCE : REAL; RATIO : INTEGER); EXTERNAL;
PROCEDURE SWEEP_TO_EDGE(MOVEMENT : CHAR; VAR TURN_ANGLE : REAL; VAR AV_RANGE : REAL); EXTERNAL;
PROCEDURE WRITE_CHARSET(S : STRING); EXTERNAL;
PROCEDURE WRITE_REAL(NMBR : REAL); EXTERNAL;
PROCEDURE WRITELN_REAL(NMBR : REAL); EXTERNAL;

PROCEDURE GET_INTER_DEST(VAR PREY_DEST : POS_PTR);

VAR
THIS_ANGLE,LEFT_ANGLE,RIGHT_ANGLE : REAL;
LEFT_RANGE,RIGHT_RANGE : REAL;
DEST : POS_PTR;
AV_RANGE,TURN_ANGLE : REAL;

BEGIN("*FIND EDGE WHICH CAUSES THE LEAST DEVIATION*")
THIS_ANGLE := ROBOT.ANGLE;
LEFT_ANGLE := ZERO;
LEFT_RANGE := ZERO;
SWEEP_TO_EDGE('L',LEFT_ANGLE,LEFT_RANGE);
MOVE_ROB('R',LEFT_ANGLE * HALF_WHEELBASE,1);
RIGHT_ANGLE := ZERO;
RIGHT_RANGE := ZERO;
SWEEP_TO_EDGE('R',RIGHT_ANGLE,RIGHT_RANGE);
IF LEFT_ANGLE < RIGHT_ANGLE THEN
BEGIN(*CALCULATE THE INTERMEDIATE DESTINATION FROM THE EDGE WHICH CAUSES THE LEAST DEVIATION*)
AV_RANGE := LEFT_RANGE;
TURN_ANGLE := LEFT_ANGLE + THIS_ANGLE + (1.15 *
ARCTAN(HALF_WHEELBASE/LEFT_RANGE));
(*THIS GIVES A LARGE MARGIN OF AVOIDANCE*)
END
ELSE
BEGIN
AV_RANGE := RIGHT_RANGE;
TURN_ANGLE := THIS_ANGLE - (RIGHT_ANGLE + (1.15 *
ARCTAN(HALF_WHEELBASE/RIGHT_RANGE)));
END;
NEW(DEST);
DEST.XPOS := ROBOT.XCOORD + (1.35*AV_RANGE*COS(TURN_ANGLE));
DEST.YPOS := ROBOT.YCOORD + (1.35*AV_RANGE*SIN(TURN_ANGLE));
(*INSERT NEW DESTINATION INTO LIST*)
WRITE_CHAR('INTER DEST ');
WRITE_REAL(DEST.XPOS);
WRITE_REAL(DEST.YPOS);
DEST.NEXT_POS := PREV_DEST.NEXT_POS;
PREV_DEST.NEXT_POS := DEST;
END;(*GET_INTER_DEST*)

8086

SPOINTER_SIZE 32S
$FAR_LIBRARIES ON$ $FAR_PROC ON$ $EXTENSIONS ON$ $DEBUG ON$ $GLOBPROC ON$ $SEPARATE CONST OFF$

PROGRAM PSWEEP_TO_EDGE;
CONST
PI = 3.14159;
WHEELBASE = 17.20;

PROCEDURE MOVE_ROB(MOVEMENT : CHAR; DISTANCE : REAL; RATIO : INTEGER);EXTERNAL;
PROCEDURE ONE_PULSE(VAR RANGE : REAL);EXTERNAL;

PROCEDURE SWEEP_TO_EDGE(MOVEMENT : CHAR; VAR TURN_ANGLE : REAL; VAR AV_RANGE : REAL); 
VAR
ANGLE_INC,DISTANCE : REAL;
TEMP_RANGE : REAL;

BEGIN
ONE_PULSE(TEMP_RANGE);
DISTANCE := 0.844;
ANGLE_INC := DISTANCE/ WHEELBASE;
(*SCAN THE OBJECT UNTIL THE EDGE IS DETECTED, SAVE THE ANGLE*)
REPEAT(*UNTIL 'STEP'*)
AV_RANGE := TEMP_RANGE;
MOVE_ROB(MOVEMENT,DISTANCE,3);
ONE_PULSE(TEMP_RANGE);
TURN_ANGLE := TURN_ANGLE + ANGLE_INC;
UNTIL(ABS(AV_RANGE - TEMP_RANGE) > 30.0) OR (TURN_ANGLE > PI);
END;(*SWEEP_TO_EDGE*)
PROGRAM FIND_PATH;

TYPE

INTP = INTEGER; (*USED WHEN INITIALISING HEAP*)
VECTOR = RECORD XCOORD,YCOORD,ANGLE : REAL END;
POS_PTR = PLACE;
PLACE = RECORD
  XPOS,YPOS : REAL;
  NEXT_POS,PREV_POS : POS_PTR;
END;

CONST

HEAPSIZE = UNSIGNED_16(0EFFH); (*HEAP SIZE OF 0EFFH BYTES*)

VAR

$EXTVAR ON$

ROBOT : VECTOR;
SIMULATION : BOOLEAN;
HEAPSTART : INTEGER; (*DEFINE BEGINING OF HEAP: IN MO86IOA*)

PROCEDURE INITHEAP(START : INT; SIZE : UNSIGNED_16);
EXTERNAL;
PROCEDURE READLN_REAL(VAR NMBR : REAL); EXTERNAL;
PROCEDURE WRITE_CHAR(S : STRING); EXTERNAL;
PROCEDURE AVOID(VAR FIRST_POS : POS_PTR;
  VAR PATH_CLEAR : BOOLEAN); EXTERNAL;

PROCEDURE FIND_PATH;

VAR

FIRST_POS,DEST : POS_PTR;
PATH_CLEAR : BOOLEAN;

BEGIN
  (*INIT HFAP*)
  INITHEAP(ADDR(HEAPSTART),HEAPSIZE);

  (*GET DESTINATION*)
  NEW(DEST);
  FIRST_POS := DEST;
  FIRST_POS.XPOS := ROBOT.XCOORD;
  FIRST_POS.YPOS := ROBOT.YCOORD;
  SIMULATION := TRUE; (*FIND PATH THROUGH MODEL ENVIRONMENT*)
  REPEAT
    ROBOT.XCOORD := FIRST_POS.XPOS;
    ROBOT.YCOORD := FIRST_POS.YPOS;
    PATH_CLEAR := TRUE;
    AVOID(FIRST_POS,PATH_CLEAR);
  UNTIL PATH_CLEAR;
  (*NOW MOVE MOBILE ROBOT THROUGH PHYSICAL ENVIRONMENT*)
  END; (*FIND_PATH*)
PROGRAM DIFFP;

TYPE
SCAN_ARRAY = ARRAY[0..99] OF REAL;
VAR
$EXTVAR ON$
RANGE_ARRAY, DIFF_ARRAY : SCAN_ARRAY;
$EXTVAR OFF$
PROCEDURE READLN_REAL(VAR NMBR : REAL); EXTERNAL;
PROCEDURE READLN_CHAR(VAR CH : CHAR); EXTERNAL;
PROCEDURE WRITE_CHAR(S : STRING); EXTERNAL;
PROCEDURE DIFF;
VAR
ANS : CHAR;
NORM_FACTOR : REAL;
INDEX : INTEGER;
BEGIN
(*DIFFERENTIATE RANGE_ARRAY*)
DIFF_ARRAY[0] := 0.0;
DIFF_ARRAY[1] := 0.0; (*AVOIDS LARGE STEP AT BEGINING*)
FOR INDEX := 2 TO 99 DO
DIFF_ARRAY[INDEX] := RANGE_ARRAY[INDEX] - RANGE_ARRAY[INDEX - 1]; (*DIFF*)
END; (*DIFF*)

PROGRAM PRE_PROCESS;

CONST
NOISE_LIMIT = 3;

TYPE
SCAN_ARRAY = ARRAY[0..99] OF REAL;
VAR
$EXTVAR ON$
DIFF_ARRAY, RANGE_ARRAY : SCAN_ARRAY;
$EXTVAR OFF$
PROCEDURE DIFF; EXTERNAL;
PROCEDURE PRE_PROCESS(POS_THRES, NEG_THRES : REAL);
VAR
INC, COUNT : REAL;
INDEX, STEP : INTEGER;
BEGIN
(*PRELIM DIFF*)
DIFF;
(*AVOID FIRST AND LAST RESULTS*)
FOR INDEX := 1 TO 99 DO
IF DIFF_ARRAY[INDEX] < NEG_THRES THEN
BEGIN
COUNT := 1.0;
REPEAT COUNT := COUNT + 1.0;
UNTIL (DIFF_ARRAY[INDEX + ROUND(COUNT)] > POS_THRES) OR
(ROUND(COUNT) + INDEX > 98);
END; (*REMOVES NOISE OF NARROW WIDTH*)
IF COUNT < NOISE_LIMIT THEN (*REMOVE FROM RANGE_ARRAY*)
BEGIN

INC := (RANGE_ARRAY[INDEX - 1] - RANGE_ARRAY[INDEX +
ROUND(COUNT)]) / COUNT;
FOR STEP := INDEX TO (INDEX + ROUND(COUNT)) DO
  RANGE_ARRAY[STEP] := RANGE_ARRAY[STEP] - INC;
END;(*IF*)
END;(*IF AND FOR*)
(*RE PRE_PROCESS WITH NOISE LIMITED DATA*)
DIFF;
END;(*PRE_PROCESS*)

"8086"
$POINTER_SIZE 32$
$FAR_LIBRARIES ON$
$FAR_PROC ON$
$EXTENSIONS ON$
$DEBUG ON$
$SEPARATE_CONST OFF$

PROGRAM DETECT_EDGES;

CONST
PI = 3.14159;
HALF_TURN = 800.0;

TYPE
SCAN_ARRAY = ARRAY[0..99] OF REAL;
EDGE_ARRAY = ARRAY[1..20,1..3] OF REAL;

VAR
$EXTVAR ON$
DIFF_ARRAY,RANGE_ARRAY : SCAN_ARRAY;
$EXTVAR OFF$

PROCEDURE GET_THRES(VAR POS : REAL;VAR NEG : REAL); EXTERNAL;
PROCEDURE PRE_PROCESS(POS_THRES,NEG_THRES : REAL); EXTERNAL;
PROCEDURE WRITE_CHAR(S : STRING); EXTERNAL;
PROCEDURE WRITELN_REAL(NMBR : REAL); EXTERNAL;
PROCEDURE READLN_CHAR(VAR CH : CHAR); EXTERNAL;
PROCEDURE DETECT_EDGES(VAR NUMB_EDGES : INTEGER;
  VAR EDGE_LIST : EDGE_ARRAY);

VAR
COUNT,ANGLE_INC : REAL;
INDEX : INTEGER;
POS_THRES,NEG_THRES : REAL;
ANS : CHAR;

BEGIN
  GET_THRES(POS_THRES,NEG_THRES);
  PRE_PROCESS(POS_THRES,NEG_THRES);(*DIFF,AND NOISE REMOVAL*)
  NUMB_EDGES := 0;
  ANGLE_INC := 8.0 * PI/HALF_TURN;(*ANGLE BETWEEN RANGES*)
  COUNT := 0.0;
  REPEAT(*UNTIL COUNT > 99.0*)
(*CHECK EACH DIFFERENTIATED RANGE VALUE FOR EXCEEDING THE
  POSITIVE OR NEGATIVE THRESHOLD, AND SAVE IN A LIST*)
INDEX := ROUND(COUNT);
IF DIFF_ARRAY[INDEX] > POS_THRES THEN
  BEGIN
    NUMB_EDGES := NUMB_EDGES + 1;
    EDGE_LIST[NUMB_EDGES,1] := 1.0;(*POSITIVE*)
    EDGE_LIST[NUMB_EDGES,2] := RANGE_ARRAY[INDEX -
2];(*RANGE*)
    (*IGNORE EXTREME EDGE VALUE OF RANGE*)
    EDGE_LIST[NUMB_EDGES,3] := (COUNT -1.0) *
    ANGLE_INC;(*ANGLE*)
    (*BUT ASSUME THAT THE ANGLE IS OK*)
  END
  ELSE
    IF DIFF_ARRAY[INDEX] < NEG_THRES THEN
      BEGIN
        NUMB_EDGES := NUMB_EDGES + 1;
        EDGE_LIST[NUMB_EDGES,1] := -1.0;(*NEGATIVE*)
        EDGE_LIST[NUMB_EDGES,2] := RANGE_ARRAY[INDEX +1];(*RANGE*)
    (*IGNORE EXTREME EDGE VALUE OF RANGE*)
    EDGE_LIST[NUMB_EDGES,3] := COUNT * ANGLE_INC;(*ANGLE*)
    (*BUT ASSUME THE ANGLE IS OK*)
    END;(*IF*)
  COUNT := COUNT + 1.0;
  UNTIL COUNT > 99.0;
(*ASK USER IF EDGES ARE TO BE DISPLAYED ON THE VDU*)
WRITE_CHAR('SEE EDGES? ');
READLN_CHAR(ANS);
IF ANS = 'Y' THEN
  FOR INDEX := 1 TO NUMB_EDGES DO
    BEGIN
      WRITE_CHAR('CLASS ');
      WRITELN_REAL(EDGE_LIST(INDEX,1));
      WRITE_CHAR('RANGE ');
      WRITELN_REAL(EDGE_LIST(INDEX,2));
      WRITE_CHAR('ANGLE ');
      WRITELN_REAL(1000.0 * EDGE_LIST(INDEX,3));
    END;(*FOR*)
  END;(*DETECT_EDGES*)

"8086"
$POINTER SIZE 32$
$FAR_LIBRARIES ON$
$FAR_PROC ON$
$EXTENSIONS ON$
$DEBUG ON$
$GLOBPROC ON$
$SEPARATE_CONST OFF$

PROGRAM GET_THRESP;

PROCEDURE WRITE_CHAR(S : STRING); EXTERNAL;
PROCEDURE READLN_CHAR(VAR CH : CHAR); EXTERNAL;
PROCEDURE READLN_REAL(VAR NMBR : REAL); EXTERNAL;

PROCEDURE GET_THRESP(VAR POS : REAL;
  VAR NEG : REAL);

VAR
  ANS : CHAR;
BEGIN(*ASK USER WHAT THRESHOLDS ARE TO BE USED*)
  WRITE_CHAR('NEW THRES? ');
  READLN_REAL(ANS);
  IF ANS = 'Y' THEN
    BEGIN(*USER DEFINED THRESHOLDS*)
    POS := 30.0;(*CM*)
    NEG := -30.0;
  END
END

ELSE
  BEGIN(*STANDARD THRESHOLDS*)
    POS := 30.0;(*CM*)
    NEG := -30.0;
  END
END;(*GET_THRESP*)
"8086"

$POINTER_SIZE 32$
$FAR_LIBRARIES ON$
$FAR_PROC ON$
$EXTENSIONS ON$
$DEBUG ON$
$SEPARATE_CONST OFF$

PROGRAM DETECT_OBJSP;
CONST
ZERO = 0.0;

TYPE
EDGE ARRAY = ARRAY[1..20,1..3] OF REAL;
OBJ ARRAY= ARRAY[1..10,1..3] OF REAL;

PROCEDURE DETECT_EDGES(VAR NUMB EDGES:INTEGER;VAR EDGE LIST:EDGE ARRAY);EXTERNAL;
PROCEDURE WRITE_REAL(NMBR : REAL); EXTERNAL;
PROCEDURE WRITELN_REAL(NMBR : REAL); EXTERNAL;
PROCEDURE WRITELN_CHAR(S : STRING); EXTERNAL;
PROCEDURE DETECT_OBJS(VAR OBJ_COUNT: INTEGER;VAR OBJ_LIST : OBJ ARRAY);

VAR
AV_RANGE,OBJ_ANGLE,SUB_ANGLE : REAL;
INDEX,NUMB EDGES : INTEGER;
EDGE_LIST : EDGE ARRAY;

BEGIN
NUMB EDGES := 0;
DETECT_EDGES(NUMB EDGES,EDGE_LIST);
OBJ_COUNT := 0;
FOR INDEX := 1 TO NUMB EDGES DO
IF (EDGE_LIST[INDEX,1] < ZERO) AND (*NEGATIVE*)
(EDGE_LIST[INDEX + 1,1] > ZERO) THEN (*POSITIVE*)
BEGIN
OBJ_COUNT := OBJ_COUNT + 1;
AV_RANGE := (EDGE_LIST[INDEX,2] + EDGE_LIST[INDEX +1,2])/2.0;
(*ANGLE SUBTENDED BY OBJECT*)
SUB_ANGLE := (EDGE_LIST[INDEX +1,3] - EDGE_LIST[INDEX,3]) /
(EDGE_LIST[INDEX,2]);
OBJ_ANGLE := (EDGE_LIST[INDEX,3] + EDGE_LIST[INDEX 
+1,3])/2.0;
OBJ_LIST[OBJ_COUNT,1] := AV_RANGE;(* + (AV_RANGE *

OBJ_LIST[OBJ_COUNT,2] := OBJ_ANGLE;(*ORIENTATION OF
OBJECT: RELATIVE*)
OBJ_LIST[OBJ_COUNT,3] := AV_RANGE * SUB_ANGLE;(*WIDTH OF
OBJECT*)
END;(*IF AND FOR*)
(*DISPLAY OBJECTS*)
WRITELN_CHAR(' RANGE ANGLE SIZE');
FOR INDEX := 1 TO OBJ_COUNT DO
BEGIN
WRITE_REAL(OBJ_LIST[INDEX,1]);
WRITE_REAL(OBJ_LIST[INDEX,2] * 1000.0);
WRITELN_REAL(OBJ_LIST[INDEX,3]);
END;
WRITELN_CHAR('FINISHED DETECTING@');
END;(*DETECT_OBJS*)

CD
PROGRAM FIND_POSITIONP;

TYPE

INTP = INTEGER; (*USED TO INIT HEAP*)
OBJ_ARRAY = ARRAY[1..10,1..3] OF REAL;
KNOWN_ARRAY = ARRAY[1..7,1..3] OF REAL;
TRI_PTR = TRI;
TRIPLE_PTR = TRIPLE;
TRI = RECORD
  FROM_OBJ,TO_OBJ : INTEGER;
  DIST : REAL;
  SEC_LEV : TRIPLE_PTR;
  NEXT_OBJ : TRI_PTR;
END;
TRIPLE = RECORD
  FIRST,SECOND : TRI_PTR;
  NEXT_TRIPLE : TRIPLE_PTR;
END;

CONST

HEAP_SIZE = UNSIGNED_16(0EFFH); (*DEFINES HEAP SIZE AS 0EFFH BYTES*)

VAR

$EXTVAR ON$;

HEAPSTART : INTEGER; (*DEFINES BEGINING OF HEAP IN MOB8610A*)

$EXTVAR OFFS$

PROCEDURE Writeln_CHAR(CH : CHAR); EXTERNAL;
PROCEDURE InitHeap(START : INTP; SIZE : UNSIGNED_16); EXTERNAL;
PROCEDURE Detect_OBJS(VAR OBJ_COUNT : INTEGER; VAR OBJ_LIST : OBJ_ARRAY); EXTERNAL;
PROCEDURE Get_Known_OBJ; EXTERNAL;
PROCEDURE Set_Up_OBJ_DATA(VAR FIST : TRI_PTR); EXTERNAL;
PROCEDURE Object_Match(NUMB_OBJ : INTEGER;
  VAR OBJ_LIST : OBJ_ARRAY;
  VAR FIRST : TRI_PTR); EXTERNAL;

PROCEDURE FIND_POSITION;
VAR

PTR,FIRST : TRI_PTR;
NUMB_OBJ : INTEGER;
OBJ_LIST : OBJ_ARRAY;

BEGIN
  InitHeap(ADDR(HEAPSTART),HEAP_SIZE);
  NEW(PTR);
  FIRST := PTR;
  FIRST.FROM_OBJ := 0; (*ESTABLISHES FIRST - I THINK!*)
  Detect_OBJS(NUMB_OBJ,OBJ_LIST); (*DETECTS ANY OBJECTS*)
  Get_Known_OBJ; (*PUTS KNOWN OBJS INTO LINKED LIST*)
  IF NUMB_OBJ > 2 THEN
    BEGIN
      Set_Up_OBJ_Data(FIRST); (*FORM KNOWN OBJ DATA STRUCTURE*)
      Object_Match(NUMB_OBJ,OBJ_LIST,FIRST); (*MATCH OBJECT DISTRIBUTION AND CALCULATION THE VALID POSITIONS*)
    END
    ELSE Writeln_CHAR('NOT 3 OBJ');
  END;(*FIND_POSITION*)
"8086"
$POINTER_SIZE 32$
$FAR_LIBRARIES ON$
$FAR_PROC ON$
$EXTENSIONS ON$
$DEBUG ON$
$GLOBPROC ON$

PROGRAM GET_KNOWN_OBJ;

CONST
MAX_KNOWN = 7;

TYPE
KNOWN_ARRAY = ARRAY[1..7,1..3] OF REAL;

VAR
$EXTVAR ON$
KNOWN_OBJ : KNOWN_ARRAY;

$EXTVAR OFF$

PROCEDURE WRITELN_REAL(NMBR : REAL); EXTERNAL;
PROCEDURE WRITE_CHAR(S : STRING); EXTERNAL;
PROCEDURE WRITE_INT(NMBR : INTEGER); EXTERNAL;
PROCEDURE WRITE_REAL(NMBR : REAL); EXTERNAL;
PROCEDURE READLN_CHAR(CH : CHAR); EXTERNAL;
FUNCTION SQR(NMBR : REAL) : REAL; EXTERNAL;
PROCEDURE INIT_KNOWN_OBJ; EXTERNAL;

PROCEDURE GET_KNOWN_OBJ;

VAR
ANS : CHAR;
NUMB_KN,INDEX,WHICH_OBJ : INTEGER;
TEMP_REAL : REAL;

BEGIN("ASK USER FOR SOURCE OF OBJ DATA")
WRITE_CHAR('ALTER OJS? ');
READLN_CHAR(ANS);
IF ANS = 'Y' THEN
BEGIN("CHANGE EXISTING OBJECTS")
WRITE_CHAR('TYPE? ');
READLN_CHAR(ANS);
CASE ANS OF
'C' : BEGIN("CHANGE SINGLES")
FOR INDEX := 1 TO MAX_KNOWN DO("DISPLAY THEM")
BEGIN
WRITE_INT(INDEX);
WRITE_REAL(KNOWN_OBJ[INDEX,1]);
WRITE_REAL(KNOWN_OBJ[INDEX,2]);
WRITE_REAL(KNOWN_OBJ[INDEX,3]);
END;(*FOR*)
ANS := 'Y';
WHILE ANS = 'Y' DO
BEGIN
WRITE_CHAR('OBJ NMBR? ');
READLN_REAL(TEMP_REAL);
WHICH_OBJ := ROUND(TEMP_REAL);
IF (WHICH_OBJ > 0) AND (WHICH_OBJ < MAX_KNOWN +
1) THEN
BEGIN
WRITE_CHAR('X ');
READLN_REAL(KNOWN_OBJ[WHICH_OBJ,1]);
WRITE_CHAR('Y ');
READLN_REAL(KNOWN_OBJ[WHICH_OBJ,2]);
KNOWN_OBJ[WHICH_OBJ,3] :=
SQRT(SQR(KNOWN_OBJ[WHICH_OBJ,1]) +
SQR(KNOWN_OBJ[WHICH_OBJ,2]));
WRITE_CHAR('MORE? ');
READLN_CHAR(ANS);
END
ELSE WRITELN_CHAR('ERR ');
END;(*CHANGE")
'N' : BEGIN("ALL NEW OBJECTS")
INIT_KNOWN_OBJ;
WRITE_CHAR('NUMBER? ');
READLN_REAL(TEMP_REAL);
NUMB_KN := ROUND(TEMP_REAL);
FOR_INDEX := 1 TO NUMB_KN DO
BEGIN
WRITE_CHAR('X ');
READLN_REAL(KNOWN_OBJ[INDEX,1]);
WRITE_CHAR('Y ');
READLN_REAL(KNOWN_OBJ[INDEX,2]);
KNOWN_OBJ[INDEX,3] :=
SQRT(SQR(KNOWN_OBJ[INDEX,1]) + SQR(KNOWN_OBJ[INDEX,2]));
END;(*FOR")
END;(*CASE")
END;(*GET_KNOWN_OBJ")

CD"
PROGRAM SET_UP_DATAP;

TYPE
TRIPLE_PTR = TRIPLE;
TRI_PTR = TRI;
TRI = RECORD
  FROM_OBJ, TO_OBJ : INTEGER;
  DIST : REAL;
  SEC_LEV : TRIPLE_PTR;
  NEXT_OBJ : TRI_PTR;
END;
TRIPLE = RECORD
  FIRST, SECOND : TRI_PTR;
  NEXT_TRIPLE : TRIPLE_PTR;
END;
OBJ_PTR = OBJECT;
OBJECT = RECORD
  WHICH_OBJ : INTEGER;
  XVAL, YVAL : REAL;
  NEXT_OBJ : OBJ_PTR;
END;

PROCEDURE SECOND_LEVEL(VAR INITIAL : TRI_PTR); EXTERNAL;

PROCEDURE SET_UP_OBJ_DATA(VAR INITIAL : TRI_PTR);

VAR
WIDTH, SEC_WIDTH : TRI_PTR;
TWOLEV : TRIPLE_PTR;
HIGHEST, LOWEST, FIRST_OBJ, SEC_OBJ : INTEGER;
TEMP_FIRST : TRI_PTR;

BEGIN(*COMPLETES THE TRIPLEPTRS KNOWN OBJ DATA STRUCTURE*)
SECOND_LEVEL(INITIAL);(*SETS UP TRIPLEPTRS*)
WIDTH := INITIAL;(*GET FIRST WIDTH*)
REPEAT(*UNTIL WIDTH = NIL*)
  TWOLEV := WIDTH.SEC_LEVEL;
  REPEAT(*UNTIL TWOLEV = NIL*)
    FIRST_OBJ := WIDTH.FROM_OBJ;(*GET BOTH OBJECT NUMBERS*)
    SEC_OBJ := WIDTH.TO_OBJ;
    TEMP_FIRST := TWOLEV.FIRST;
  UNTIL TWOLEV = NIL;
  SEC_WIDTH := INITIAL;(*MATCH HIGHEST NUMBER*)
  WHILE SEC_WIDTH.FROM_OBJ <> HIGHEST DO
    SEC_WIDTH := SEC_WIDTH.NEXT_OBJ;
  END;
  WHILE SEC_WIDTH.TO_OBJ <> LOWEST DO(*MATCH LOWEST NUMBER*)
    SEC_WIDTH := SEC_WIDTH.NEXT_OBJ;
    TWOLEV.SECOND := SEC_WIDTH;
  END;
  WIDTH := WIDTH.NEXT_OBJ;
  UNTIL WIDTH = NIL;
  TWOLEV := TWOLEV.NEXT_TRIPLE;
UNTIL TWOLEV = NIL;
END;(*SET_UP_OBJ_DATA*)

WITH TEMP_FIRST DO
  IF FIRST_OBJ = FROM_OBJ THEN FIRST_OBJ := TO_OBJ
  ELSE IF FIRST_OBJ = TO_OBJ THEN FIRST_OBJ := FROM_OBJ
  ELSE IF SEC_OBJ = FROM_OBJ THEN SEC_OBJ := TO_OBJ
  ELSE SEC_OBJ := FROM_OBJ;(*WITH*)
  IF FIRST_OBJ > SEC_OBJ THEN
    BEGIN
      HIGHEST := FIRST_OBJ;
      LOWEST := SEC_OBJ;
      END;
  ELSE BEGIN
    HIGHEST := SEC_OBJ;
    LOWEST := FIRST_OBJ;
    END;
    SEC_WIDTH := INITIAL;(*MATCH HIGHEST NUMBER*)
    WHILE SEC_WIDTH.FROM_OBJ <> HIGHEST DO
      SEC_WIDTH := SEC_WIDTH.NEXT_OBJ;
    END;
    WHILE SEC_WIDTH.TO_OBJ <> LOWEST DO(*MATCH LOWEST NUMBER*)
      SEC_WIDTH := SEC_WIDTH.NEXT_OBJ;
      TWOLEV.SECOND := SEC_WIDTH;
    END;
    TWOLEV := TWOLEV.NEXT_TRIPLE;
    UNTIL TWOLEV = NIL;
    WIDTH := WIDTH.NEXT_OBJ;
    UNTIL WIDTH = NIL;
    END;(*SET_UP_OBJ_DATA*)

CDE"
PROGRAM OBJ_DATAP;

TYPE

OBJ_PTR = OBJECT;
OBJECT = RECORD
            WHICH_OBJ : INTEGER;
            XVAL, YVAL : REAL;
            NEXT_OBJ : OBJ_PTR;
        END;

KNOWN_ARRAY = ARRAY[1..7,1..3] OF REAL;

CONST

MAX_KNOWN = 7;

VAR

KNOWN_OBJ : KNOWN_ARRAY;

PROCEDURE OBJ_DATA(VAR FIRST_OBJ : OBJ_PTR;
                   VAR NUMB_OBJ : INTEGER);

VAR

TEMP_OBJ : OBJ_PTR;
WHICH : INTEGER;

BEGIN(*FORMS OBJECT DATA INTO A SIMPLE LINKED LIST*)

FIRST_OBJ := NIL;
NUMB_OBJ := 0;
FOR WHICH := 1 TO MAX_KNOWN DO
BEGIN
    NUMB_OBJ := NUMB_OBJ + 1;
    NEW(TEMP_OBJ);
    WITH TEMP_OBJ DO
    BEGIN
        XVAL := KNOWN_OBJ[WHICH,1];
        YVAL := KNOWN_OBJ[WHICH,2];
    END;
    TEMPOBJ.NEXT_OBJ := FIRST_OBJ;
    FIRST_OBJ := TEMP_OBJ;
END;(*FOR*)
END;(*OBJ_DATA*)

PROGRAM CALC_DISTSP;

TYPE

TRIPLE_PTR = TRIPLE;
TRI_PTR = TRI;
TRI = RECORD
  FROM_OBJ, TO_OBJ : INTEGER;
  DIST : REAL;
  SECLEV : TRIPLE_PTR;
  NEXT_OBJ : TRI_PTR;
END;

TRIPLE = RECORD
  FIRST, SECOND : TRI_PTR;
  NEXT_TRIPLE : TRIPLE_PTR;
END;

OBJ_PTR = OBJECT;
OBJECT = RECORD
  WHICH_OBJ : INTEGER;
  XVAL, YVAL : REAL;
  NEXT_OBJ : OBJ_PTR;
END;

PROCEDURE OBJ_DATA(VAR FIRST_OBJ : OBJ_PTR; VAR NUMB_OBJ : INTEGER); EXTERNAL;
FUNCTION SQR(NMBR : REAL): REAL; EXTERNAL;

PROCEDURE CALC_DIST(VAR FIRST : TRI_PTR);

VAR

NUMB_OBJ, ONE_OBJ, TWO_OBJ : INTEGER;
FIRST_OBJ, ONE_PTR, TWO_PTR : OBJ_PTR;
LAST, PREV, WIDTH : TRI_PTR;

BEGIN

(*GET OBJECT DATA INTO A SIMPLE LINKED LIST*)
OBJ_DATA(FIRST_OBJ, NUMB_OBJ);
LAST := NIL;(*SET UP LINKED LIST*)
NEW(PREV);
PREV.NEXT_OBJ := LAST;
FIRST := PREV;
ONE_OBJ := NUMB_OBJ;

ONE_PTR := FIRST_OBJ;
REPEAT (*UNTIL ONE_OBJ <= 1*)
  TWO_OBJ := ONE_OBJ - 1;
  TWO_PTR := ONE_PTR.NEXT_OBJ;
  REPEAT (*UNTIL TWO_OBJ <= ZERO*)
  (*CALCULATE THE DISTANCES BETWEEN OBJECTS AND FORM A LINKED LIST*)
  NEW(WIDTH);
  WIDTH.FROM_OBJ := ONE_OBJ;
  WIDTH.TO_OBJ := TWO_OBJ;
  WIDTH.DIST := SQRT(SQR(ONE_PTR.XVAL - TWO_PTR.XVAL) +
                   SQR(ONE_PTR.YVAL - TWO_PTR.YVAL));
  WIDTH.NEXT_OBJ := LAST;
  PREV.NEXT_OBJ := WIDTH;
  PREV := WIDTH;
  TWO_OBJ := TWO_OBJ - 1;
  TWO_PTR := TWO_PTR.NEXT_OBJ;
  UNTIL TWO_OBJ < -1;
  ONE_OBJ := TWO_OBJ - 1;
  ONE_PTR := ONE_PTR.NEXT_OBJ;
  UNTIL ONE_OBJ < -2;
  FIRST := FIRST.NEXT_OBJ;(*REMOVE DUMMY RECORD*)
END;(*CALC_DIST*)
.
"8086"

`$POINTER_SIZE 32$
$FAR_LIBRARIES ON$
$FAR_PROC ON$
$EXTENSIONS ON$
$DEBUG ON$
$GLOBPROC ON$
$SEPARATE_CONST OFF$

PROGRAM SECOND_LEVELP;

TYPE

TRIPLE_PTR = TRIPLE;
TRI_PTR = TRI;
TRI = RECORD
  FROM_OBJ, TO_OBJ : INTEGER;
  DIST : REAL;
  SEC_LEV : TRIPLE_PTR;
  NEXT_OBJ : TRI_PTR;
END;

TRIPLE = RECORD
  FIRST, SECOND : TRI_PTR;
  NEXT_TRIPLE : TRIPLE_PTR;
END;

OBJ_PTR = OBJECT;
OBJECT = RECORD
  WHICH_OBJ : INTEGER;
  XVAL, YVAL : REAL;
  NEXT_OBJ : OBJ_PTR;
END;

PROCEDURE CALC_DISTS(VAR FIRST : TRI_PTR); EXTERNAL;
PROCEDURE SECOND_LEVEL(VAR INITIAL : TRI_PTR);

VAR

WIDTH, SEC_WIDTH : TRI_PTR;
TWO_LEV : TRIPLE_PTR;

BEGIN
  (*SET UP FIRST LEVEL OF LINKS*)
  CALC_DISTS(INITIAL);
  (*NOW MAKE SECOND LEVEL LINKS*)
  WIDTH := INITIAL;
  REPEAT (*UNTIL WIDTH = NIL*)
  BEGIN
    SEC_WIDTH := INITIAL; (*POINT TO START OF LIST*)
    WIDTH.SEC_LEV := NIL;
    REPEAT (*UNTIL SEC_WIDTH = NIL*)
    BEGIN
      IF ((WIDTH.FROM_OBJ <> SEC_WIDTH.FROM_OBJ) OR
          (WIDTH.TO_OBJ <> SEC_WIDTH.TO_OBJ)) AND
          \n          (WIDTH.FROM_OBJ = SEC_WIDTH.FROM_OBJ) OR
          (WIDTH.TO_OBJ = SEC_WIDTH.TO_OBJ)) OR
          ((WIDTH.FROM_OBJ = SEC_WIDTH.FROM_OBJ) OR
          (WIDTH.TO_OBJ = SEC_WIDTH.TO_OBJ)) THEN
          \n          NEW(TWO_LEV);
          TWO_LEV.FIRST := SEC_WIDTH;
          TWO_LEV.NEXT_TRIPLE := WIDTH.SEC_LEV;
          WIDTH.SEC_LEV := TWO_LEV;
          END;
        SEC_WIDTH := SEC_WIDTH.NEXT_OBJ; (*GET NEXT ON LIST*)
        UNTIL SEC_WIDTH = NIL;
        WIDTH := WIDTH.NEXT_OBJ; (*GET NEXT WIDTH*)
        UNTIL WIDTH = NIL;
      END; (*SECOND_LEVEL*)
  END; (*SECOND_LEVEL*)
PROGRAM OBJECT_MATCHP;

TYPE
OBJ_ARRAY = ARRAY[1..10,1..3] OF REAL;
KNOWN_ARRAY = ARRAY[1..7,1..3] OF REAL;
TRI_PTR = TRI;
TRIPLE_PTR = TRIPLE;

TRI = RECORD
  FROM_OBJ, TO_OBJ : INTEGER;
  DIST : REAL;
  SECLEV : TRIPLE_PTR;
  NEXT_OBJ : TRI_PTR;
END;
TRIPLE = RECORD
  FIRST, SECOND : TRI_PTR;
  NEXT_TRIPLE : TRIPLE_PTR;
END;

PROCEDURE WRITE_CHAR(S : STRING); EXTERNAL;
PROCEDURE WRITE_INT(NMBR : INTEGER); EXTERNAL;
PROCEDURE WRITE_REAL(NMBR : REAL); EXTERNAL;
PROCEDURE WRITELN_REAL(NMBR : REAL); EXTERNAL;
PROCEDURE WRITELN_INT(NMBR : INTEGER); EXTERNAL;
PROCEDURE WRITELN_CHAR(VAR CH : CHAR); EXTERNAL;
PROCEDURE READLN_REAL(VAR NMBR : REAL); EXTERNAL;
FUNCTION SQR(NMBR : REAL); EXTERNAL;
PROCEDURE MATCH_SECOND_LEVEL(VAR WIDTH : TRI_PTR;
  VAR OBJ_LIST : OBJ_ARRAY;
  ONE, TWO, THREE : INTEGER);

PROCEDURE OBJECT_MATCH(NUMB_OBJ : INTEGER;
  VAR OBJ_LIST : OBJ_ARRAY;
  VAR FIRST : TRI_PTR);

VAR
  DIST12, DIST13, DIST23 : REAL;
  ONE, TWO, THREE : INTEGER;
  WIDTH : TRI_PTR;
  STOP_MATCH : BOOLEAN;
  INDEX : INTEGER;
  TEMP_REAL : REAL;
  TEMP_CHAR : CHAR;

BEGIN
(*DISPLAY DETECTED OBJECTS*)
FOR INDEX := 1 TO NUMB_OBJ DO
  BEGIN
    WRITE_INT(INDEX);
    WRITE_REAL(OBJ_LIST[INDEX,1]);
    WRITE_REAL(OBJ_LIST[INDEX,2] * 1000.0);
    WRITE_REAL(OBJ_LIST[INDEX,3]);
  END;
END;

(*USE THE CO-SINE RULE TO FIND DISTANCES APART*)
DIST12 := COS(ABS(OBJ_LIST[ONE,2] - OBJ_LIST[TWO,2]));
DIST12 := DIST12 * OBJ_LIST[TWO,1];
DIST12 := DIST12 * 2.0 * OBJ_LIST[ONE,1];
TEMP_REAL := SQR(OBJ_LIST[ONE,1]) + SQR(OBJ_LIST[TWO,1]);
DIST12 := SQRT(TEMP_REAL - DIST12);
DIST13 := COS(ABS(OBJ_LIST[ONE,2] - OBJ_LIST[THREE,2]));
DIST13 := DIST13 * OBJ_LIST[THREE,1];
DIST13 := DIST13 * 2.0 * OBJ_LIST[ONE,1];
TEMP_REAL := SQR(OBJ_LIST[ONE,1]) + SQR(OBJ_LIST[THREE,1]);
DIST13 := SQRT(TEMP_REAL - DIST13);
DIST23 := COS(ABS(OBJ_LIST[TWO,2] - OBJ_LIST[THREE,2]));
DIST23 := DIST23 * OBJ_LIST[THREE,1];
DIST23 := DIST23 * 2.0 * OBJ_LIST[TWO,1];
TEMP_REAL := SQR(OBJ_LIST[TWO,1]) + SQR(OBJ_LIST[THREE,1]);
DIST23 := SQRT(TEMP_REAL - DIST23);
WRITE_REAL(DIST12);
WRITE_REAL(DIST13);
WRITELN_REAL(DIST23);
WIDTH := WIDTH.NEXT_OBJ;
REPEAT(*COMPARE DIST BETWEEN A DETECTED OBJECT PAIR AND KNOWN OBJECT PAIRS*)
  IF ABS(DIST12 - WIDTH.DIST) < DIST12/10.0 THEN
    (*WHEN A MATCH IS FOUND CHECK THE REMAINING TWO DistANCES*)
    MATCH_SECOND_LEVEL(WIDTH,DIST23,DIST13,
      OBJ_LIST,ONE,TWO,THREE);
  END;
UNTIL WIDTH = NIL;
END;(*FOR LOOP*)
END;(*OBJECT_MATCH*)
PROGRAM MATCH_SEC_LEVP;

TYPE

OBJ_ARRAY = ARRAY[1..10,1..3] OF REAL;
KNOWN_ARRAY = ARRAY[1..7,1..3] OF REAL;
TRI_PTR = TRI;
TRIPLE_PTR = TRIPLE;

TRI = RECORD
  FROM_OBJ, TO_OBJ : INTEGER;
  DIST : REAL;
  SEC_LEV : TRIPLE_PTR;
  NEXT_OBJ : TRI_PTR;
END;

TRIPLE = RECORD
  FIRST, SECOND : TRI_PTR;
  NEXT_TRIPLE : TRIPLE_PTR;
END;

PROCEDURE WRITE_CHAR(S : STRING); EXTERNAL;

PROCEDURE CALC_POS(VAR WIDTH : TRI_PTR;
  VAR SEC_LEVEL : TRIPLE_PTR;
  VAR OBJ_LIST : OBJ ARRAY;
  ONE, TWO, THREE : INTEGER); EXTERNAL;

PROCEDURE MATCH_SECOND_LEVEL(VAR WIDTH : TRI_PTR;
  DIST23, DIST13 : REAL;
  VAR OBJ_LIST : OBJ ARRAY;
  ONE, TWO, THREE : INTEGER);

VAR

SEC_LEVEL : TRIPLE_PTR;
TEMP_REAL : REAL;
TEMP_TRI : TRI_PTR;

BEGIN
  SEC_LEVEL := WIDTH.SEC_LEV;
  REPEAT(*UNTIL SEC_LEVEL = NIL*)
  (*COMPARE DISTANCES BETWEEN SECOND AND THIRD OBJECTS*)
  (*IF ALL THREE MATCH CALCULATE THE POSITION OF THE MOBILE ROBOT*)

  TEMP_TRI := SEC_LEVEL.FIRST;
  TEMP_REAL := ABS(DIST23 - TEMP_TRI.DIST);

  IF TEMP_REAL < (DIST23/10.0) THEN
    BEGIN(*COMPARE THE THIRD DISTANCE*)
      TEMP_TRI := SEC_LEVEL.SECOND;
      IF (ABS(DIST13 - TEMP_TRI.DIST) < (DIST13/10.0)) THEN
      (*IF*)
        SEC_LEVEL := SEC_LEVEL.NEXT_TRIPLE;
      UNTIL SEC_LEVEL = NIL;
    END;(*IF*)
  END;(*MATCH_SECOND_LEVEL*)
PROGRAM CALC_POS;

TYPE

OBJ_ARRAY = ARRAY[1..10,1..3] OF REAL;
KNOWN_ARRAY = ARRAY[1..7,1..3] OF REAL;
TRIPLE_PTR = TRIPLE;

TRI = RECORD
  FROM_OBJ, TO_OBJ : INTEGER;
  SEC_LVL : TRIPLE_PTR;
END;

TRIPLE = RECORD
  FIRST, SECOND : TRI_PTR;
END;

VAR

KNOWN_OBJ : KNOWN_ARRAY;

PROCEDURE WRITE_REAL(NMBR : REAL; EXTERNAL);
PROCEDURE WRITELN_REAL(NMBR : REAL; EXTERNAL);
PROCEDURE WRITE_INT(NMBR : INTEGER; EXTERNAL);
PROCEDURE WRITELN_INT(NMBR : INTEGER; EXTERNAL);
FUNCTION SQ(R : REAL) : REAL; EXTERNAL;
PROCEDURE CALC_POS(VAR WIDTH : TRI_PTR;
VAR OBJ_LIST : OBJ_ARRAY;
VAR ONE, TWO, THREE : INTEGER);

VAR

TEMP1, TEMP2, TEMP3 : REAL;
DIVISOR, X_VALUE, Y_VALUE : REAL;

BEGIN
  "ASSIGN FIRST SECOND AND THIRD OBJECTS"
  WITH WIDTH DO
    IF (FROM_OBJ = SEC_LVL.FIRST.FROM_OBJ OR
        FROM_OBJ = SEC_LVL.FIRST.TO_OBJ) THEN
      BEGIN
        FIRST := TO_OBJ;
        SECOND := FROM_OBJ;
      END
    ELSE BEGIN
      FIRST := FROM_OBJ;
      SECOND := TO_OBJ;
    END;
  END;

  WITH SEC_LVL.FIRST DO
    IF SECOND = FROM_OBJ THEN THIRD := TO_OBJ
    ELSE THIRD := FROM_OBJ;
  END;

  WRITE_INT(FIRST); WRITE_INT(SECOND); WRITE_INT(THIRD);

  (GET THE 3 SELECTED RANGES FROM THE DETECTED OJBS LIST)
  IF ((KNOWN_OBJ[FIRST,3] < 1.0) OR (KNOWN_OBJ[SECOND,3] < 1.0) OR
      (KNOWN_OBJ[THIRD,3] < 1.0)) THEN WRITELN_CHAR('CANNOT FIND POS')
  (*TEST FOR CORRECTLY SELECTED OBJECTS*)
  ELSE BEGIN
    (*TEMPI := SQR(KNOWN_OBJ(SECOND,3) + SQR(OBJ_LIST[THREE,1]) -
      SQR(KNOWN_OBJ[THIRD,3]) - SQR(OBJ_LIST[TWO,1]);
    TEMP2 := SQR(OBJ_LIST[ONE,1]) + SQR(KNOWN_OBJ[THIRD,3]) +
      SQR(KNOWN_OBJ[FIRST,3]) + SQR(OBJ_LIST[THREE,1]);
    TEMP3 := TEMP1 - (SQR(KNOWN_OBJ[THIRD,3]) +
      SQR(OBJ_LIST[ONE,1]));
    TEMP1 := SQR(KNOWN_OBJ[SECOND,3]) - SQR(OBJ_LIST[THREE,1]) -
      SQR(OBJ_LIST[TWO,1]);
    TEMP2 := TEMP2 - (SQR(KNOWN_OBJ[SECOND,3]));
    TEMPI := (KNOWN_OBJ[SECOND,3] - FROM_OBJ) * (KNOWN_OBJ[SECOND,2] -
      KNOWN_OBJ[THIRD,2]) - (KNOWN_OBJ[FIRST,3] - FROM_OBJ) * (KNOWN_OBJ[FIRST,2] -
      KNOWN_OBJ[THIRD,2]);
    TEMP1 := TEMP1 - (KNOWN_OBJ[THIRD,3]) - (KNOWN_OBJ[FIRST,3]) -
      (KNOWN_OBJ[FIRST,2];
    TEMP1 := TEMP1 - (KNOWN_OBJ[SECOND,3]);
    TEMP1 := TEMP1 - (KNOWN_OBJ[FIRST,3]);
    TEMP1 := TEMP1 - (KNOWN_OBJ[SECOND,3]);
    TEMP1 := TEMP1 - (KNOWN_OBJ[FIRST,3]);
  END;

  (*THESE SYMBOLS MATCH THE ALGORITHM AND CALCULATE THE X AND Y
   COORDINATES OF THE POSITION NOTE: THE ORIENTATION IS NOT FOUND*)

  TEMP1 := SQR(KNOWN_OBJ[SECOND,3]) + SQR(OBJ_LIST[THREE,1]) -
    SQR(KNOWN_OBJ[THIRD,3]) - SQR(OBJ_LIST[TWO,1]);
  TEMP2 := SQR(OBJ_LIST[ONE,1]) + SQR(KNOWN_OBJ[THIRD,3]) +
    SQR(KNOWN_OBJ[FIRST,3]) + SQR(OBJ_LIST[THREE,1]);
  TEMP3 := TEMP1 - (SQR(KNOWN_OBJ[THIRD,3]) +
    SQR(OBJ_LIST[ONE,1]));
  TEMP1 := SQR(KNOWN_OBJ[SECOND,3]) - SQR(OBJ_LIST[THREE,1]) -
    SQR(OBJ_LIST[TWO,1]);
  TEMP2 := TEMP2 - (SQR(KNOWN_OBJ[FIRST,3])) +
    SQR(KNOWN_OBJ[SECOND,3]);
  TEMP1 := TEMP1 - (SQR(KNOWN_OBJ[THIRD,3]) +
    SQR(OBJ_LIST[ONE,1]));

  DIVISOR := (KNOWN_OBJ[SECOND,1] - KNOWN_OBJ[FIRST,1]) *
    (KNOWN_OBJ[SECOND,2] - KNOWN_OBJ[THIRD,2]);
  TEMP_DIV := (KNOWN_OBJ[THIRD,1] - KNOWN_OBJ[FIRST,1]) *
    (KNOWN_OBJ[FIRST,2] - KNOWN_OBJ[SECOND,2]);
  DIVISOR := 2.0 * (DIVISOR - TEMP_DIV) + 1.0;
  X_VALUE := (KNOWN_OBJ[SECOND,2] * TEMP1) +
    (KNOWN_OBJ[SECOND,2] * TEMP2);
X.VALUE := (X.VALUE + (KNOWN_OBJ[THIRD,2] * TEMP3)) / DIVISOR;
DIVISOR := DIVISOR-1 (X.VALUE + (KNOWN_OBJ[SECOND,1] * TEMP2));
Y.VALUE := (Y.VALUE + (KNOWN_OBJ[THIRD,1] * TEMP3)) / DIVISOR;
WRITE_CHAR('X ');
WRITE_REAL(X.VALUE);
WRITE_CHAR('Y ');
WRITELN_REAL(Y.VALUE);
END;(*IF*)
END;(*CALC_POS*)

"8086"
$POINTER SIZE 32$ $FAR LIBRARIES ON$ $FAR-PROC ON$
$EXTENSIONS ON$ $DEBUG ON$ $GLOBPROC ON$
$SEPARATE_CONST OFF$

PROGRAM PSET_UP_WORLD;

TYPE
LINE_ARRAY = ARRAY[0..64,0..3] OF REAL;

VAR
$GLOBVAR ON$
WORLD : LINE_ARRAY;
NUMBLINES : INTEGER;

$GLOBVAR OFF$
PROCEDURE WRITELN_CHAR(S : STRING); EXTERNAL;
PROCEDURE WRITE_CHAR(S : STRING); EXTERNAL;
PROCEDURE READLN_CHAR(VAR CH : CHAR); EXTERNAL;
PROCEDURE READLN_INT(VAR NMBR : INTEGER); EXTERNAL;
PROCEDURE READLN_REAL(VAR NMBR : REAL); EXTERNAL;
PROCEDURE WRITE_REAL(NMBR : REAL); EXTERNAL;
PROCEDURE WRITE_REAL(NMBR : REAL); EXTERNAL;
PROCEDURE WRITE_REAL(NMBR : REAL); EXTERNAL;
PROCEDURE WRITE_REAL(NMBR : REAL); EXTERNAL;
PROCEDURE WRITE_REAL(NMBR : REAL); EXTERNAL;
PROCEDURE WRITE_REAL(NMBR : REAL); EXTERNAL;
PROCEDURE WRITE_REAL(NMBR : REAL); EXTERNAL;

PROCEDURE SET_UP_WORLD;

VAR
INDEX : INTEGER;
ANSWER : CHAR;

BEGIN(*ASK USER IF NEW OR OLD OBJS ARE REQUIRED*)
WRITELN_CHAR('NEW OBJECTS? ');
READLN_CHAR(ANSWER);
IF ANSWER = 'Y' THEN
BEGIN(*ENTER NEW OBJS AS LINE SEGMENTS*)
WRITELN_CHAR('FORMAT IS X1,Y1,X2,Y2');
FOR INDEX := 1 TO NUMBLINES DO
BEGIN
WRITELN_CHAR('NEXT SIDE VERTICES');
READLN_REAL(WORLD[INDEX,0]);
READLN_REAL(WORLD[INDEX,1]);
READLN_REAL(WORLD[INDEX,2]);
READLN_REAL(WORLD[INDEX,3]);
END;(*FOR*)
ELSE
BEGIN(*MAKE CHANGES TO EXISTING LINE SEGMENTS*)
FOR INDEX := 1 TO NUMBLINES DO
BEGIN
WRITE_INT(INDEX);
WRITE_REAL(WORLD[INDEX,0]);
WRITE_REAL(WORLD[INDEX,1]);
WRITE_REAL(WORLD[INDEX,2]);
WRITELN_REAL(WORLD[INDEX,3]);
END;(*FOR*)
WRITE_CHAR('WHICH LINE? ');
READLN_INT(INDEX);
READ_REAL(WORLD[INDEX,0]);
READ_REAL(WORLD[INDEX,1]);
READ_REAL(WORLD[INDEX,2]);
READLN_REAL(WORLD[INDEX,3]);
(*THE SIMULATED ENVIRONMENT IS NOW READY TO USE*)
END;(*IF*)
END;(*SET_UP_WORLD*)

"8086"
$POINTER_SIZE 32S
$FAR_LIBRARIES ON
$FAR_PROC ON
$EXTENSIONS ON
$DEBUG ON
$GLOBPROC ON
PROGRAM INIT_KNOWN_OBJ;
TYPE
KNOWN_ARRAY = ARRAY[1..7,1..3] OF REAL;
CONST
MAX_KNOWN = 7;
VAR
$EXTVAR ON
KNOWN_OBJ : KNOWN_ARRAY;
$EXTVAR OFFS
PROCEDURE INIT_KNOWN_OBJ;
VAR
INDEX : INTEGER;
BEGIN
(*CLEAR ARRAY : THIS CAN BE DETECTED IN CALC_POS*)
FOR INDEX := 1 TO MAX_KNOWN DO
BEGIN
KNOWN_OBJ[INDEX,1] := 0.0;(*X-COOR*)
KNOWN_OBJ[INDEX,2] := 0.0;(*Y-COOR*)
KNOWN_OBJ[INDEX,3] := 0.0;(*DIST FROM ORIGIN*)
END;(*FOR*)
END;(*INIT_KNOWN_OBJ*)
"8086"

$POINTER SIZE 32$
$FAR_LIBRARIES ON$
$FAR_PROC ON$
$EXTENSIONS ON$
$DEBUG ON$
$GLOBPROC ON$
$SEPARATE_CONST OFF$

PROGRAM MOVE_GETP;

CONST HALF_TURN = 54.0;

TYPE VECTOR = RECORD XCOOR, YCOOR, ANGLE : REAL END;

VAR

$EXTVAR ON$

ROBOT VECTOR;

$EXTVAR OFF$

PROCEDURE WRITE_CHAR(S : STRING); EXTERNAL;
PROCEDURE READLN_CHAR(VAR CH : CHAR); EXTERNAL;
PROCEDURE READLN_REAL(VAR NMBR : REAL); EXTERNAL;
PROCEDURE MOVE_ROB(MOVEMENT : CHAR; REAL_DIST : REAL; RATIO : INTEGER); EXTERNAL;
PROCEDURE MOVE_GET(RATIO : INTEGER);

VAR

FINISHED : BOOLEAN;
MOVEMENT : CHAR;
REAL_DIST : REAL;

BEGIN/*MOVE THE PROTOTYPE MOBILE ROBOT UNDER PROGRAM CONTROL*/
FINISHED := FALSE;
REPEAT /*UNTIL FINISHED*/
  WRITE_CHAR('ENTER DIRECTION ');
  READLN_CHAR(MOVEMENT);
  CASE MOVEMENT OF
    'F', 'B' : BEGIN
      WRITE_CHAR('DISTANCE IN CM ');
      READLN_REAL(REAL_DIST);
      MOVE_ROB(MOVEMENT, REAL_DIST, RATIO);
    END;
    'L', 'R' : BEGIN
      WRITE_CHAR('ANGLE IN DEGREES ');
      READLN_REAL(REAL_DIST);
      REAL_DIST := REAL_DIST * HALF_TURN/180.0;
    END;
  OTHERWISE FINISHED := TRUE;
END;/*CASE*/
UNTIL FINISHED;
END;/*MOVE_GET*/
"8086"

$POINTER_SIZE 32$
$FAR_LIBRARIES ON$
$FAR_PROC ON$
$EXTENSIONS ON$
$DEBUG ON$
$GLOBPROC ON$
$SEPARATE_CONST OFF$

PROGRAM DISPLAY_POSP;

TYPE

VECTOR = RECORD XCOORD, YCOORD, ANGLE : REAL END;

CONST

PI = 3.14159;
TWO_PI = 6.283185308;
ZERO = 0.0;

PROCEDURE WRITECHAR(S : STRING) ; EXTERNAL;
PROCEDURE WRITE_REAL(NMBR : REAL) ; EXTERNAL;
PROCEDURE DISPLAY_POSITION(POS : VECTOR);

BEGIN
  WITH POS DO
    BEGIN
      WRITECHAR('XCOORD');
      WRITE_REAL(XCOORD/10.0);
      WRITECHAR('YCOORD');
      WRITE_REAL(YCOORD/10.0);
      WRITECHAR('ANGLE');
      WRITE_REAL(ANGLE * 180.0/PI);
    END; (*WITH*)
END; (*DISPLAY_POSITION*)

END; (*DISPLAY_POSP*)
PROCEDURE PUTCHAR(CH : CHAR);
BEGIN
  CHAROUT := CH;
  OUTCHAR;
END; (*PUTCHAR*)

PROCEDURE WRITE_CHAR(S : STRING);
VAR INDEX : INTEGER;
BEGIN
  FOR INDEX := 1 TO ORD(S[0]) DO
    PUTCHAR(S[INDEX]);
  END; (*WRITE_CHAR*)

PROCEDURE WRITELN_CHAR(S : STRING);
VAR INDEX : INTEGER;
BEGIN
  FOR INDEX := 1 TO ORD(S[0]) DO
    PUTCHAR(S[INDEX]);
  LINEFEED;
END; (*WRITELN_CHAR*)

PROCEDURE READ_REAL(VAR NUMBER : REAL);
UNTIL ENTERED;
IF SIGN THEN NUMBER := 0.0 - NUMBER;
NUMBER := NUMBER/10.0;
END; (*READ_REAL*)

PROCEDURE READLN_REAL(VAR NMBR : REAL);
BEGIN
  READ_REAL(NMBR);
  LINEFEED;
END; (*READLN_REAL*)

PROCEDURE READLN_INT(VAR NMBR : INTEGER);
VAR REAL NMBR : REAL;
BEGIN
  READLN_REAL(REAL NMBR);
  NMBR := ROUND(REAL NMBR);
END; (*READLN_INT*)

PROCEDURE WRITE_INT(NMBR : INTEGER);
VAR POWERCOUNT, VALUE, TENPOWER : INTEGER;
STARTED : BOOLEAN;
BEGIN
  IF NMBR < 0 THEN
    BEGIN
      WRITE_CHAR('-');
      NMBR := 0 - NMBR;
    END
  ELSE WRITE_CHAR(' ');
  TENPOWER := 10000;
  POWERCOUNT := 0;
  STARTED := FALSE;
  REPEAT (*UNTIL POWERCOUNT > 4*)
  VALUE := 0;
  WHILE NMBR >= TENPOWER DO
    BEGIN
      NMBR := NMBR - TENPOWER;
      VALUE := VALUE + 1;
    END;
  IF VALUE > 0 THEN STARTED := TRUE;
  CASE VALUE OF
  0 : IF STARTED OR (POWERCOUNT > 3) THEN PUTCHAR(' 0') ELSE
    PUTCHAR(' ');
  1 : PUTCHAR('1');
  2 : PUTCHAR('2');
  3 : PUTCHAR('3');
  4 : PUTCHAR('4');
  5 : PUTCHAR('5');
  6 : PUTCHAR('6');
begin
    writeln_int(nmb);
    writeln(nmb);
    writeln(' ');
end; (*write_int*)

procedure writeln_real(nmreal : real);
begin
    write_int(round(nmreal));
    writeln(nmreal);
    writeln(' ');
end; (*write_real*)

function sqr(nmb : real) : real;
begin
    sqr := nmb * nmb;
end; (*sqr*)
"8086"

ASSEMBLY LISTING OF THE I/O ROUTINES FOR THE SDK-86 BOARD
OUTCHAR AND INCHAR COMMUNICATE TO THE VDU (RS232C) FOR
PASCAL

GLB OUTCHAR, INCHAR, CHAROUT, CHARIN
GLB DELAY, INIT_8251
GLB SEND, RECEIVE
GLB CON_SIGNED_8
GLB ADDRESS, DSTACK, NUMBYTES
GLB BITE, OVER_BITE
GLB RANGE ARRAY, DIFF_ARRAY
GLB NORM_ARRAY
GLB INIT_INTERRUPTS

OUTCHAR PROC
PUSH AX
WAITTX MOV AX, STATPORT
IN AL, AX
TEST AL, TXR(1Y
JZ WAITTX
; WAIT FOR READY
MOV DX, DATAPORT
MOV AL, CHAROUT
PUSH AX
POP BX
RET

************

INIT_INTERRUPTS: PROC
PUSH AX
; INIT THE ARITH INTERUPT LOCATIONS
MOV BX, OFFSET Underflow
MOV DS: WORD PTR 00H, BX
; UNDERFLOW ERROR
MOV BX, SEG Underflow
MOV DS: WORD PTR 02H, BX
MOV BX, OFFSET Overflow
MOV DS: WORD PTR 01H, BX
; OVERFLOW ERROR
MOV BX, SEG Overflow
MOV DS: WORD PTR 012H, BX
POP BX

************

OUTCHAR PROC
PUSH AX
WAITTX MOV AX, DX, STATPORT
IN AX AL
TEST AL, DX
JZ WAITTX
MOV AX, DX, , DATAPORT
OUT AX AL
AL: CHAROUT
OUT DX,AL
POP DX
RET

*******************************************************************************
INCHAR PROC FAR
PUSH DX
WAITRX MOV DX,STATPORT ;GET FOR I/P
IN AL,DX
TEST AL,RXRDY JZ WAITRX
MOV DX,DATAPORT
IN AL,DX
AND AL,ASMASK
MOV CHARIN,AL ;SAVE CHAR
MOV CHAROUT,AL
CMP AL,ASCR
JE ECHO IF ( = CR
JB NOTASCII CALL POP RET FAR PTR OUTCHAR ;OUTPUT CHAR
*******************************************************************************
CON_SIGNED_8 PROC FAR
;INTI 8251
PUSH AX ;SAVE AX
SUB AL,AL
MOV OVER_BITE,AL
MOV AL,BITE
AND AL,800H ;SAVE MSB
JZ MSB_ZERO
MOV AL,B1H
MOV OVER_BITE,AL
MOV AL,BITE ;MASK OFF MSB
AND AL,07FH
MOV BITE,AL
POP AX ;RESTORE AX
RET

*******************************************************************************
DELAY PROC FAR ;SHORT DELAY
NOP
NOP
NOP
NOP
NOP
RET

*******************************************************************************
INIT_8251 PROC FAR ;INTI 8251
PUSH DX ;SAVE DX
MOV DX,SICONT ;RESET OF 8251
MOV AL,SIRESET
OUT DX,AL
CALL FAR PTR DELAY ;SETTLING DELAY
*******************************************************************************
SEND PROC FAR ; SENDS VIA COMM
PUSH DI ;SAVE DI AND DX
PUSH DX
MOV DX,P2CONT ;WAIT UNTIL BUS IS CLEAR
MOV AL,RESETBUS
OUT DX,AL
BUSACTIV:
MOV DX,PORT2C
IN AL,DX
XOR AL,0FFH ;INVERT INPUT
JNE BUSACTIV
MOV DX,P2CONT
OUT DX,AL
MOV AL,OUTCONT
MOV DX,PORT2C
OUT DX,AL
MOV AL,BUSSEND
XOR AL,0FFH
OUT DX,AL
MOV DX,PORT2B
OUT DX,AL
MOV AL,ADDRESS
OUT DX,AL
MOV DX,PORT2C ;USE ACKNOW TO SIGNAL DATA STAB
WAITACT:
IN AL,DX ;WAIT FOR REC
XOR AL,0FFH
JZ WAITACT
MOV AL,00FH
XOR AL,0FFH
OUT DX,AL
MOV DX,PORT2C
OUT DX,AL
XOR AL,0FFH
JZ WAITACT
MOV DI,00H
MOV CH,00H ;SET UP COUNTER
XOR AL,THISADDR  ;IS THIS DEST?
JNZ AGAIN  ;NO, THEN WAIT
IN AL,DX  ;YES, SAVE ADDR
XOR AL,0'FFH
MOV ADDRESS,AL
MOV DX,P2CONT  ;SET UP BUS FOR
MOV AL,INCONT  ;INPUT
OUT DX,AL
MOV DX,PORT2C  ;ACTIVATE REC
XOR AL,0'FFH
OUT DX,AL
MOV DI,0'0'0'0'H  ;INIT DATA
IN AL,DX  ;COUNTER
XOR AL,0'FFH
AND AL,ACKNOWL
JNZ DATAIN  ;MORE DATA?
IN AL,DX
XOR AL,0'FFH
AND AL,ACKCLR
JZ ENDREC  ;NO, THEN EXIT
MOV DX,PORT2B  ;YES GET IT
IN AL,DX
XOR AL,0'FFH
MOV BYTE PTR DSTACK[DI],AL
INC DI  ;NEXT LOCATION
MOV DX,PORT2C  ;IN STACK
XOR AL,ACTREC
MOV AL,ACTREC
XOR AL,0'FFH
OUT DX,AL
XOR AL,0'FFH
AND AL,ACKNOWL
JZ WAITACK2
MOV AL,BUSREC  ;CLR REC
OUT DX,AL
JMP DATAIN
MOV DX,P2CONT  ;RESET BUS
MOV AL,RESETBUS
OUT DX,AL
POP DX
POP DI
RET

******************************************************************************
END
APPENDIX C

THE PROTOTYPE MOBILE ROBOT

The control process of the prototype mobile robot was performed by a distributed network of single chip microprocessors connected by a parallel bus (see figure C.1), and consisted of a chassis (see figure C.2) which contained two step motors, short and long range ultrasonic rangefinders and a joystick control.

The two main controllers were the long range ultrasonic rangefinder (see figures C.3 and C.4) and the drive motor controller for the two step motors (see figures C.5, C.6, C.7 and C.8) which were fully integrated into the system, and were coordinated by the 16 bit 8086 microprocessor on the SDK-86 which was executing the PASCAL program (see appendix B and chapter four).

The long range rangefinder measured the distance to the nearest object in line-of-sight and transmitted that value (in cm) to the SDK-86 via the parallel communication bus. The drive motor controller moved the chassis in four different directions, forwards, backwards and stationary rotation left or right, by the number of steps specified. When a movement was completed an acknowledgement was sent to the SDK-86 via the parallel communication bus (see figure C.9).

The two remaining functions, the short range obstacle detector (see figures C.10 and C.11) and the joystick controller (see figure C.12) were operational but had not been integrated into the control system. The obstacle detector was quiescent until an object was detected within 50cm of the prototype mobile robot when an alarm signal was sent to the SDK-86 via the parallel communication bus. The
Figure C.1 The distributed multiprocessor controller
Figure C.2 The mobile robot chassis
Figure C.3 Connection of the analogue PCB for the long range ultrasonic rangefinder
Figure C.4 Long-range ultrasonic rangefinder controller
Figure C.5  Connection of step motor power amplifiers
Inter processor communication bus

8 bit data bus

4 bit control bus

Motor Controller

8 bits

Power Amplifier

4 bits

Left step motor

4 bits

Right step motor

Figure C.6 Connection of motor controller for the prototype mobile robot
Figure 2.7  Step motor drive controller
Figure C.8  Power amplifier for step motors
Figure C.9 Connection of the SDK-86 to the mobile robot controller

50 Way ribbon cable connection to the SDK-86 plug J5
Figure C.10 Short range obstacle detector (ultrasonic) controller
Figure C.11 Analogue processing for the short range obstacle detector
Figure C.12 Joystick controller
joystick controller issued commands similar to those from the SDK-86 when movements were to be made, which allowed the prototype mobile robot to be manoeuvred manually. Unfortunately, as this controller was not fully integrated it could not be installed at the same time as the SDK-86, reducing its usefulness.

The Communication Protocol

The system used a simple parallel bus consisting of 8 bits of data and 4 control lines. Two of the control lines were used by any module to obtain control of the bus, and the remaining two to 'handshake' data between devices. Each separate controller was identified by an address between 1 and 255, known as the destination address as this was always the first byte of data of a message package.

The bus signals were "low" when not in use. When a device was ready to send data it checked the BUS ACTIVE (SENDER and RECEIVER) signals to see if the bus was free. When it was free the BUS ACTIVE (SENDER) and ACKNOWLEDGE control lines were activated, and the address of the receiving device placed on the data bus. When the address was stable the ACKNOWLEDGE was removed. As all module periodically scanned the control bus looking for messages the required module eventually checked the address on the data bus. It then activated BUS ACTIVE (RECEIVER) signal to let the sending device know that it was ready. The transmitting device then placed the first byte of data on the bus. The receiver detected that, saved the data and activated the RECEIVED signal. A normal handshake then took place to complete the transfer. The process was repeated if more than one data byte was transmitted. After the last data byte had been transmitted and the handshake completed, the sender removed the BUS ACTIVE (SENDER) signal. The receiver detected this and cleared the BUS ACTIVE (RECEIVER) signal, leaving the bus free.
The bus arbitration was not daisy-chained in order to add or remove controllers without affecting the remainder of the system.

A bus failure could occur at the moment of bus seizure in the period between a device scanning the control bus and finding it empty, and activating the BUS ACTIVE (SENDER) signal. This was because the bus was completely under software control and there was a period of several microseconds between the scan and action, during which time another controller could scan the bus and find it apparently unused. So far this has not been a problem. The bus protocol was used because it required no extra hardware and any updates and alterations to the protocol were easily implemented. Recovery from some errors was possible as each bus seizure had a time-out period and also the received data was checked for consistancy.

The bus protocol had no ultimate master as each controller was essentially independent and could function apart from all others. The bus simply allowed commands and sometimes data, to be passed among controllers to coordinate a complex function. Each controller had the capability to assume control of the mobile robot and send commands and receive data from all other modules.

All interaction with other modules occurred via the communication bus and apart from this, there were no direct signal connections between any controllers.