1991

Cost effective navigation of mobile robots

David Sydney Colless
University of Wollongong

Cost Effective Navigation of Mobile Robots

A thesis submitted in fulfilment of the requirments

for the award of the degree of

MASTER OF ENGINEERING

(HONOURS)

from

The University of Wollongong

by
David Sydney Colless

BAppSc Computing
Queensland Institute of Technology(1981)

BMath (Hons)
The University of Wollongong(1984)

Department of Electrical and

Computer Engineering

February 1991

Abstract

The navigation of a robot from a source to a target location is a process that requires the gathering and use of environmental information. This process is costly in terms of both sensor equipment and computer processing power.

This thesis attempts to minimise these costs by looking at minimal environment information. This is to be achieved using readily available components and computer algorithms designed to make the most of the information collected. A simple implementation of Dead Reckoning has been employed to demonstrate the gains made.

Significant savings can be achieved using these methods.

1. Introduction

1.1 Aims

This project sets out to develop a mobile platform capable of point to point navigation within strict cost constraints. This is achieved using readily available components and computer algorithms designed to make the most of the information available.

A simple method of navigation, mainly Dead Reckoning is to be used. This should be implemented with a minimal amount of analogue hardware, the emphasis should instead be on processing sensory data using computer algorithms. To achieve low cost, simple sensors must be employed detecting primitive environmental inputs. The environmental inputs to be measured are, the ability to detect when the robot is facing north, measure the distance travelled by the robot and measure the distance from an object directly in front of the robot.

1.2 Configuration

The platform selected is similar in configuration to that used by a majority of research efforts in robot navigation.

Lof and Chen[17] used a similar design based on a SDK-51 controller (8051 CPU with extra I/O ports). They developed an interpretive command language to control their platform similar to that used here. Their command language allowed the creation of loop constructs and contained extra motor control commands. Karayma and Yuta[13] also used a similar configuration but had the platform control performed by the navigation computer. Both platforms used Pulse driven motors to control the direction and speed of the robot. This configuration greatly simplifies the algorithm necessary to implement dead reckoning as part of this navigation strategy relies on being able to perform precise on the spot turns.

An alternative configuration is the use of a steerable wheel. An example of this was used by Da Casta[5]. He was unable to implement dead reckoning. Steer [22] as recently as 1989 was developing

algorithms to solve the problem of calculating the current position after performing a turn using a steerable wheel platform.

1.3 Sensors

The type of sensors used governs the environmental information available to the robot when navigating. A wide variety of sensors are in use in navigation research. The more sophisticated efforts combine the results of multiple sensors. Kriegman, Triedl and Binford[14] developed a platform using vision and several other sensors. Koch, Yeh, Hillel, Meyste and Isik[16] developed a storage system to take advantage of the information obtained via vision sensors. At the other end of the scale Everett and Flyn[7] wasted valuable information by using Near Infra-red proximity sensors simply as a means of collision avoidance.

Sensors most similar to those used in this research are laser range finders. Harmon[9] used a laser range finder to complement grey scale vision in his research. This type of sensor returns distance information in one direction only. To construct a 'Map' of the environment this type of sensor must be rotated in a scanning fashion and the map built up from the range and angle information.

The sensor used is similar in configuration to a unit that was developed commercially by Visitronics a division of Honeywell (Ref Everett[6]). The commercial unit claimed to have better performance with a range of 2 metres and a precision of better than 10mm.

1.4 Processing Power

The use of multiple CPUs is common in much of the research in this area [9][17]. The alternative is the use of a single large CPU. Kriegman, Triedl and Binford[14] used a VAX with a DMA channel dedicated to the collection of the video information. The use of such a physically large computer meant that it had to be stationary and a communication channel to the robot be provided.

The use of multiple CPUs poses a different communication problem. Harmon[9] used an IEEE 802.3 LAN to interconnect his processors. This solution was viable due to the number and variety of sensor CPUs used. They included proximity sensors, grey scale vision, laser range finders and colour vision sub systems.

The network chosen in this research was a simple RS232 line using Xon/Xoff protocol to connect the 8031 transport and sensor computer to the IBM P.C. mapping and path planning computer. This network is similar to that used by Lof and Chew[17]. Lof and Chew used an 8031 controller for the transport system with the sensor processing, mapping and path planning on the main processor.

1.5 Position Verification.

Many methods of position verification and correction are available each with different sensor and environmental information requirements. The higher the degree of accuracy required the more stringent the requirements for environmental data.

Kabuka and Arenas[12] proposed a system of standard markers. The markers contained a pattern the vision system was able to distinguish form other objects in the range of vision. The robot used dead reckoning to navigate. By comparing the relative positions of markers to the recorded positions the system was able to correct or verify the position of the vehicle at each point.

Koch, Yeh, Hillel, Meystel and Isik[16] used a more processing intensive system with video sensors and a complex data storage technique to record descriptions of the objects within the environment. By identifying and comparing the position of known objects within an environment the robot's position could be verified.

The system used has simpler optics and storage requirements than both these approaches. It avoids the need to mark the environment by storing environment features at each point and route information between points. Instead of describing objects by their vertex, only walls are considered and represented by their face angle to

north and the length of visible wall. This simplification greatly reduces the processing power and storage structures requirements compared to Kabuka[12] or Koch, Yeh, Hillel, Meystel and Isik[16].

1.6 Map Storage

As the robot was to operate in a known environment several storage schemes are possible.

Frywell[8] and Zalinski[25] have developed the Quad Tree storage technique. This method allows the storage of large amounts of environmental data in a relative small space. This storage technique requires complex path planning algorithms before path leg commands for the transport CPU can be generated.

Chatita and Laumond[13] proposed a storage technique based on a cell structure. The principle was to break each navigable area up into smaller cells until simple squares and triangles are left. This approach has the benefits of simplifying the shortest path search but would complicate the position verification algorithm used here.

The method used is perhaps most similar to that used by Siy[23]. Siy's method is similar to a road map where all intersections are at right angles. The map could be stored in a simple array of go and no go streets. The chosen storage technique is a series of connected vectors containing angle and distances, complemented with scan maps of the local environment at each intersection.

1.7 Path Planning and Obstacle Avoidance.

The process of planning the route to be taken in this system is as simple as adding inter point distances together and finding the shortest. No deviation from the pre-set paths is permitted.

The more sophisticated systems map the environment completely allowing the robot to navigate to previously unvisited locations. The ability to plan paths that run close to objects increases the risk of collision. Several common collision avoidance schemes used include

expanding the robot from a point to a barrel that completely encompasses the robot. Another popular method is to shrink the environment (move all walls in and expand objects) by half the size of the robot [3][21][22]. The method chosen is usually the one which is easiest to implement considering the storage technique used. The method used in this system avoids the need for known object avoidance altogether by only storing paths that are safe.

1.8 Implementation

To implement Dead Reckoning, a vector map of the paths through which the robot can travel was used. The map contained enough information to enable commands to be generated and issued to the drive motor controller to move the robot through the selected path.

The environment map stores information about all the known paths in the environment. A search algorithm was developed to select the shortest path from start to finish. To provide some tolerance to dynamic environments, the algorithm is re-applied when a leg of the current journey is blocked. This does not change the stored map but temporarily detours the problem location.

To achieve a high degree of error correction, as many synchronization points as practical are used. This increases the time taken to navigate, as the robot spends more time making corrections, but the net result can be significantly better.

A scan map of the local environment was created using the infra-red sensor to measure the distance from the robot to adjacent walls as the robot rotated through 360 degrees. This distance was measured by transmitting a beam of infra-red light and detecting the returned beam's relative strength. This information was converted to a digital value, smoothed and combined with direction data to give a scan map of the local environment.

All of the environment information was pre-processed before transmission to the PC. This meant that the PC can request information about the robot's current heading, distance travelled,

distance to the wall on the current heading and scan results through simple commands to the robot.

The signature of a flat wall was established to provide a method of interpreting the infra-red scan results. By searching for this "wall signature pattern" in the scan results a summary, consisting of a list of walls with their distance and angle from the robot, can be generated.

To correct the robot's position to a known synchronization point the stored scan of the current position is interpreted and compared to the expected scan at the synchronization point. The results are compared by comparing the distance to walls at similar angles. The distance differences are accumulated into an error vector.

The navigation from the start location to a target location was a repetitive process of compute path, move to first synchronization point, scan current location, compare current location with stored expected location, compute error vector and correct position according to the error vector. This process is repeated until the target location is achieved.

1.9 Results

A platform capable of navigating from point to point has been developed. It was able to achieve the aims within the required constraints but several environmental assumptions had to be added. These include :

- the need to have an environment that only contains smooth floors and flat walls that will reflect light,
- It should only operate in spaces no more than 3 metres across and
- The ability to control error is dependant on the distance between recognisable features i.e., many synchronization points give the best results.

2. Design Aims

2.1 Position Correction

The concept of dead reckoning relies on knowing the environment and being able to make precise moves within that environment. The known environment aspect is simple to supply and can be achieved by measuring and recording the environment. The ability to make precise movements within an environment is a more difficult requirement to fulfil. The two factors that need to be controlled are distance and bearing. Neither of these can be measured with any high degree of accuracy on the test vehicle. To improve this situation would incur higher costs contrary to the set objectives of the project.

The alternative is to correct the position of the platform back to that used when mapping the environment. To do this the robot needs to be able to calculate an error vector that describes the difference between the current and expected positions. The error vector should be in a form which is readily usable by the robot to correct its position. It should contain the distance and angle of the error.

Once the error has been calculated the correction process can take one of two options. The first is to immediately move to the corrected position. The second is to include components of the error vector in the move to the next known point.

2.1.1 Wall Detection

The basis of comparing points is to find differences between the expected and current environments. The comparison will be made by matching features and recording the differences. The simplest features to detect are walls. Any straight line found in an environment will be assumed to be a wall.

The environment compare algorithm will require a method of locating walls within a scan. This method should find all the walls in a scan and categorise them. The walls found should be recorded by

bearing from the robot, distance to the robot, probability that the object is a wall and the length of the wall that is seen.

2.1.2 Point Compare

Using the data refined from the current and expected scans the error analysis can be performed. It will use the bearing to the walls to match like walls in the two environments. By comparing the change in distance to matched walls the position displacement can be determined.

To provide some tolerance to errors in the environment data the wall probability factor should be used to select the walls to be used in the displacement calculations. Ordering the walls in decreasing order of confidence and selecting the most certain will improve the accuracy of the resultant error vector.

2.2 Robot Sensors.

One of the objectives of this thesis project is to minimise costs. To achieve this only simple sensors are used. This requirement limits the sensors to those readily available.

As limited processing power is available (once more due to cost constraints) simple sensors that require little pre-processing are used. Sensors must also be selected that will provide the maximum information about the environment for the cost.

2.2.1 View

To support the wall detection and point comparison process an ability to measure distances to objects is required. The requirement should be implemented using sensors that allow the robot to remain stationary and measure the distances to all localised objects. The

sensor should return the distance in a unit that is simple to convert to a standard unit of measure (i.e., mm).

The scan results should require a minimum amount of storage space. Economy of storage space is required as scan results recorded in the mapping phase will be used by the point compare facility in the navigation phase. Scan results will be stored in the map, both in memory and on disk.

The sensor should have a range between 1800mm and 3600mm. This range will enable it to "see" objects in most standard size rooms. The sensor should provide data of high enough quality so as to enable the wall detection function to differentiate walls from other objects.

2.2.2 Distance Measurement.

The ability to measure the distance travelled by the robot is essential. It will be used in both Map creation and the Navigation Phases.

Conversion from metric to the robot's unit of measure is necessary for the Mapping Phase. The operator specifies path lengths in millimetres and the map editor converts them to the robot's unit of measure. The distance can then be stored in the map and used to test paths and maps for accuracy.

In the Navigation Phase distances stored in the map will be used to control the travel of the robot. The required distance will be specified by the Navigator program and the robot controller will move the robot forwards until the required distance has been travelled.

2.2.3 Direction

Direction information will be stored with each path of the map. During the Navigation phase the angle will be passed to the controller which must turn the robot to the desired direction (as the robot can

only move forward or backward, forward being the preferred direction of motion).

Direction information will form the key to compare the environment scans. Using North as a starting direction for all scans the distances to walls can be compared independently without the need to mask other objects/features out.

The results of a scan comparison will be an error vector. This vector will be used to make corrections to the position of the robot. The vector will include direction information so once again direction control of the robot will be necessary.

A high degree of accuracy will be required in the control of the robot's direction. This control will become critical if long paths are to be used between synchronization points on a map as a small angle propagates into a large positional error.

The current direction should be maintained by the robot controller. As the compass sensor can only be read when facing north this will require the controller to keep track of the turns made by the robot since north was last seen.

2.3 Map Storage

The navigation of this platform is through a known environment. At no time does the robot go "Exploring" its environment. This means that the robot's World must be entered by the operator. To simplify the edit functions the internal representation of the environment should closely resemble the environment being represented.

Further reasons for using a storage structure similar to the environment being represented lie in the programming effort. Speed of navigating through the map by the Navigator program will never be of concern when compared to the speed of movement of the robot. It is more important that the structure be easy to understand. A structure that is simple to use and understand allows the programmer to concentrate on the Navigation problem.

The requirements of map storage are easier to understand in terms of the real world. If the robot's representation is similar to the real world then the conversions from storage requirements to program code will be as simple as possible.

The basis of position correction is the comparison of the expected (stored) local scan with the current local scan. To implement position correction it will be necessary to provide storage of scan results. This storage must last from when the map is first created by the system operator (in map edit phase) until the point is reached (in the navigation phase).

The robot's world will consist of a series of synchronization points connected by paths. These paths will have to be searched in several different ways. In the Edit phase a series of points grouped into a room should be accessible as a logical unit. It is likely that the system operator would want to work out all the points and the links between points in a room before moving to the next room.

In the Navigation phase the Navigator program is not concerned with arbitrary room labelling of points but instead only with the paths between points. It needs to search the paths in order to find the shortest journey for the robot to travel.

The two views of the same map further complicates the structure required to store the maps. A further requirement for the map storage is that permanent storage is required. This is necessary as the map Edit phase is separate from the Navigation phase. It is also desirable to be able to create several maps of different environments that can be selected and loaded whenever the robot is placed in an environment.

No map should be static, the transition from map edit to navigate phase should not be a one way path. Once a map has been tried in the navigation phase and additions are required it should require no effort to switch back to the Edit phase.

3. ASSUMPTIONS

3.1 Flat Walls

The success of the error correction system depends on the ability of the robot's infra-red sensor to detect straight walls. The first assumption was that the robot's environment should contain predominantly straight walls with few objects that have shapes other than flat sides. Furthermore, the walls should be reasonably reflective. They need not all be white but matt black should be avoided. Highly reflective wall surfaces should also be avoided as they appear closer than they are.

3.2 Smooth Floor

The robot's drive motors are driven near to capacity. This means that floor coverings that are soft will stall the robot's motors. The robot was of 'Turtle' design using two small drive wheels. This means that holes and bumps can not be easily traversed. It was therefore assumed that all floor surfaces should be smooth and hard.

3.3 No Magnetic Fields.

An integral part of both the dead reckoning and position correction systems is the ability to find north, from which to deduce all other bearings. Magnetic fields will reduce the accuracy of the compass and therefore all subsequent calculations. It was assumed that the robot would not need to operate through strong magnetic fields.

3.4 Known Environments.

The route planning algorithm works by searching the possible paths to find the shortest route before the robot starts to move. To avoid the complexities of exploring the environment it is assumed the environment is known in advance and is stored in the environment map.

3.5 Dynamics

As the environment must be known, the performance of the robot is dependant on the accuracy of this information. This also means that the performance is very sensitive to changes like walls moving, foreign objects placed in the environment and paths being blocked. The worst case is where a wall moves which results in the correction process positioning the robot off a synchronization point. This error may be corrected at the next synchronization point or, if the induced error is large enough, correction may not be possible.

A further aspect of environmental dynamics is due to the relatively slow infra-red scan speed. The imposed limitation here is that nothing (within range) may change during a scan. The impact of such intrusion is the appearance of walls where none exist with the possible side effects previously described.

3.6 Test Vehicle specific

The robot software was developed specifically for the test platform. The lack of software portability is attributable to the speed of the drive motors being matched to that of the micro-controller. The design presented here will require some change to account for robot speed and micro processor performance. It was assumed the software would only have to run on the test platform.

4. DESIGN

4.1 SUMMARY

The robot's initial configuration consisted of a Tandy 1000 Laptop computer connected to a 8031 micro-controller. The micro-controller had a control card[19][20] attached capable of controlling the two main motors. The software supplied with the 8031 was solely for the control of the two drive motors. Source code was not provided.

The Tandy 100 Laptop is a memory based machine which uses part of its memory as a disk system. A maximum of 64k of memory is possible in which to store and run programs and store data. It soon proved to be too small to store the programs required to implement the navigation and edit functions. Due to these limitations the decision was made to move to a PC. An attempt was made to port the software which had been written on the Laptop to the PC. This was complicated by the change of programming language from Basic to C[15][18] and as a result the only code that was reused was the Shortest Path [10] search module (Appendix A.2.3).

The software for the 8031[11] micro-controller had to be rewritten to support the compass and infra-red sensors and to increase the precision of the wheel movements.

To support the scan function an interface to the infra-red sensor was required. The output of the infra-red sensor is a signal voltage between 0.2 and 4.7 volts which is proportional to the distance an object is from the muzzle of the sensor. This interface was accomplished via an Analog to Digital converter configured as a memory location on the 8031. To provide pre-processing of the scan results extended memory had to be added to the 8031. This memory is used to store the scan results during the scan and to provide a work area to perform the smoothing routine.

The compass interface uses two of the available bits on one of the 8031 ports. One bit was configured via software for output to control the deflection electromagnet. The second bit was configured as an input from the compass's needle detection sensor.

To increase the accuracy of the robot's movements brakes were fitted. They were of simple design consisting of a solenoid connected to a lever that comes in contact with the drive wheels when the solenoid is energised. The control for the brake solenoids was via a bit on one of the 8031's ports.

The robot is controlled from a stationary PC via an umbilical cord. This cord provides RS232 communications to the robot as well as its power supply. To limit the weight of the cord a single 12 volt supply is used. The use of a single power rail means that a regulator capable of supplying +/- 12 volts and +5 volts to the robot is required. The performance of the regulator is critical as the +/- 12 volt supply is used by the operational amplifiers in the infra-red sensor, the RS232 communications drivers as well as the main drive motors.

The final hardware/software configuration is as below:

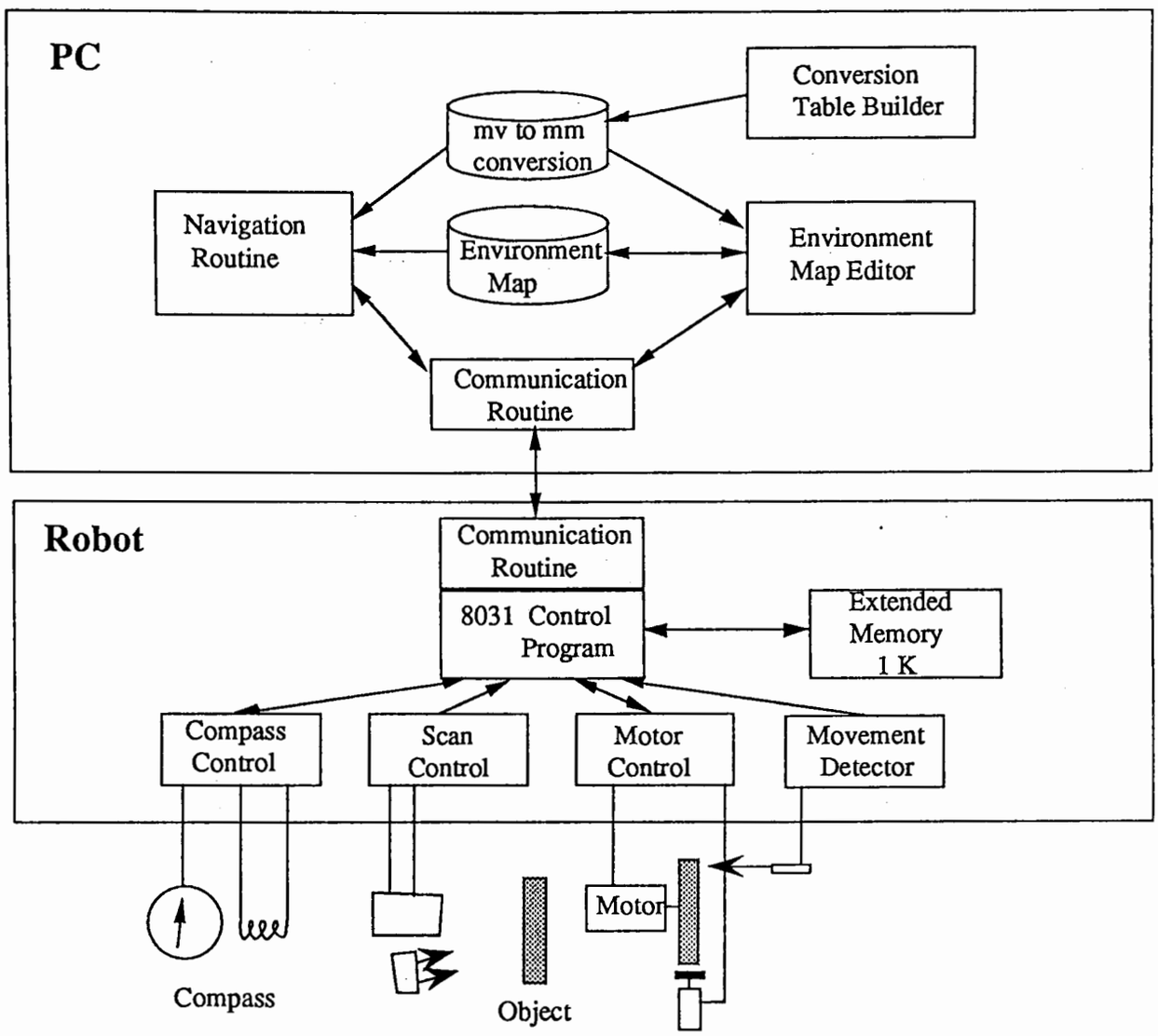


Fig 4.1 Robot Configuration

4.2 NAVIGATION

4.2.1 SUPPORT FUNCTIONS

The kernel of the navigation program is concerned with just 3 tasks: map search, position correction and commanding the robot to follow the selected path. These functions cannot occur without several support functions. Firstly, the navigation program must have the robot

in a known state, ready to accept commands. It must then load into memory a map file of the current environment. Finally, it must be able to find the robot's current position on the map and select a target location on the map. One other function has been added for demonstration purposes. This function displays details of each point visited and the correction actions taken during a navigation run.

4.2.1.1 Robot Initialisation

The first support function is used to initialise the robot. This function works by setting up the PC's serial port to the protocol used by the robot's micro-controller serial port. It then sends a reset command to the robot. The reset command initialises the counters and timing constants in the micro-controller's on-chip memory. It then tests the extended memory and turns the robot to face north. The sequence is completed by sending a status response line back to the PC.

4.2.1.2 Environment Map Load

One of the design requirements of the navigation program is that the environment map is known in advance. So before any navigation can occur the environment map must be loaded from disc. This function prompts the operator for the file name of an environment map. It then uses the load procedure described in permanent map storage to read the file, allocate memory and store the structures necessary for the navigation program to operate.

4.2.1.3 Select Start and Finish Points

Once an environment map has been loaded from disc the current relative position of the robot in the map must be determined. This requires the operator to place the robot on a synchronization point and select that point on the map. To do this a list of available rooms is displayed. The operator selects the required room and a list of points for that room is then displayed. Once the point matching the robot's

current position is selected its memory address is recorded for use by the search program.

The object of the system is to navigate from point A to point B. Once the start location (point A) has been selected, the target (point B) can be selected. This is done using the same room and point display screens used in selecting the robot's current position in the environment map. Once the target point is selected its memory address is stored for use by the search program.

To save time in the navigation process, once an environment map is loaded, the current position selected and the target position nominated the search program is initiated. If a route is found it is stored for later use.

4.2.1.4 Single Step

The single step function falls more into the demonstration category than the navigation support. It displays details about the actions being performed and the accuracy of the moves taken. Once the end of that step is complete the program returns to the main menu.

This function calls the navigation routine once for every leg of a journey. As the navigation routine instructs the robot to move, the distance, angle and name of the point at the end of the path are displayed. Once the point is reached, the correction procedure is called and the correction requirements are displayed.

This function was useful in testing the navigation routine and can be used to demonstrate how the navigation system (dead reckoning and position correction) works.

4.2.2 PATH SEARCH

The route search routine scans the current environment map to find the shortest route from the current location to a selected target

position. It uses a search algorithm called Depth First[10]. To do this it needs access to the current environment map and the start and finish points. The routine was written to search the environment map and return a list of paths. This list is then used by the move procedure to generate commands to move the robot to the target location.

The algorithm used is very simple and is included in Appendix A.2.3. It works by keeping two lists of points describing the route to the target. The first list is the current list. This list records where the search is at the current time. It also records the cost (in distance) of this route. The second list contain the best route to the target found so far. The cost of the best route is also recorded.

As the search progresses paths are added to the current list by exploring every path leading away from the current point. As a path is explored the point it leads to becomes the current point and the length of the path is added to the current cost. If the current cost exceeds the best cost then that path of the current route is discarded and the search back tracks to the previous point and subtracts the path cost from the current cost.

A check is also made at each point that the point is not included in the current route already. This has been simplified by including a flag on the point record that indicates if the point is on the current list. If the point is flagged as on the current list it is ignored and the search back tracks. As the search retreats from points it has included in the current list, these points are flagged as not in use. The in use check is necessary to prevent the search algorithm looping.

Once the target location is found if the current cost is less than the best cost the best list is replaced with the current. Once all paths from each point are searched the route in the best list is the optimum path.

An example of this back tracking is described in figure 4.2.

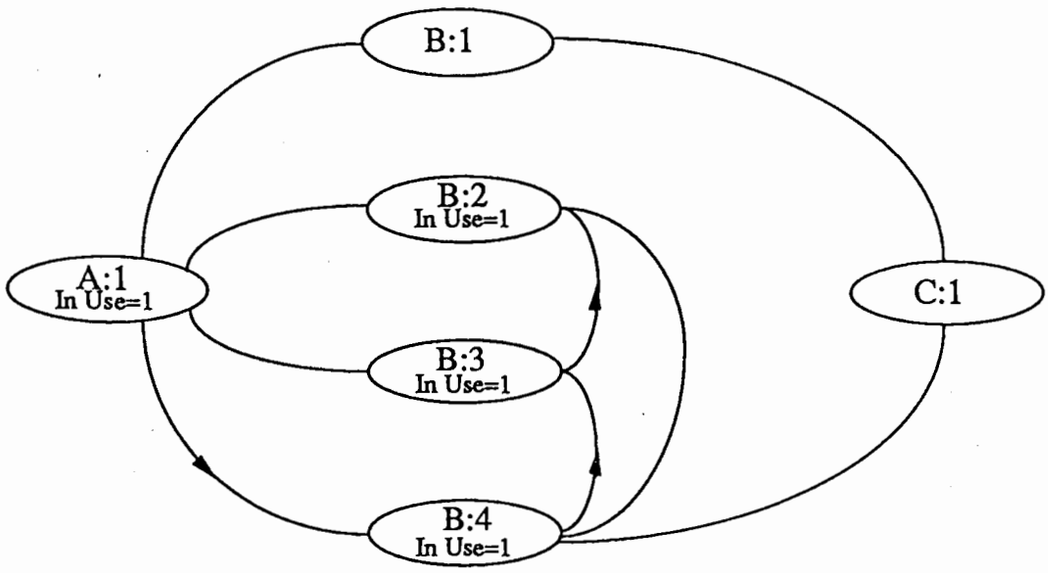


Fig 4.2 Example Search Pattern

The search has reached point B:2. The current route contains the following paths A:1->B:4, B:4->B:3 and B:3->B:2. At B:2 the search retreats as the 3 paths that lead from B:2 all lead to points (i.e., A:1, B:3 and B:4) that are flagged as in use.

The search will retreat to B:4 from where it will investigate B:4->C:1. As this point is the target, the current route (A:1->B:4,B:4->C:1) will replace the best route, providing the route cost is less than the best route cost.

4.2.3 COMPARE POINTS

The problem here is to compare two scan results to give a distance and angle representing the offset from the robot to the expected position. This process is not a straight forward comparison of the scans, observation by observation. To compare scan observations one at a time would ignore too much information stored in the combined observations that make up a scan.

The information stored in a infra-red scan contains details of local environment features. These include the placement of objects in the room as well as the walls of the room. As the available processing power is limited, straight object (like walls) only were considered.

To identify straight line objects requires the definition of a pattern that can be used to locate these objects within a scan. As it turns out such a pattern is simple to define and a method of identifying the pattern is simple to implement. Once all the walls of a room have been found they can be compared. This was done by finding the right angle displacement to each wall in the stored scan and comparing it with the matching wall in the current scan to select an error vector.

4.2.4 DETECT WALLS

A pattern that represents straight lines in a scan was found using the following method. The distance from a point to a straight line follows the formula

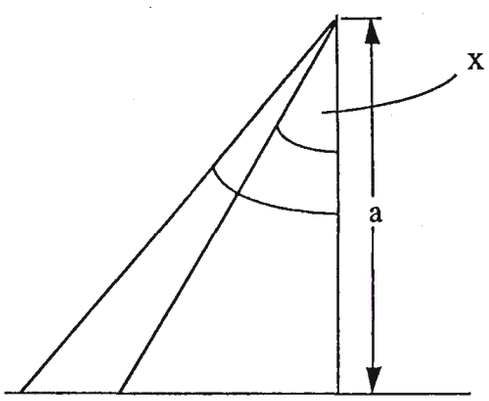


Fig 4.3 Distance to a wall

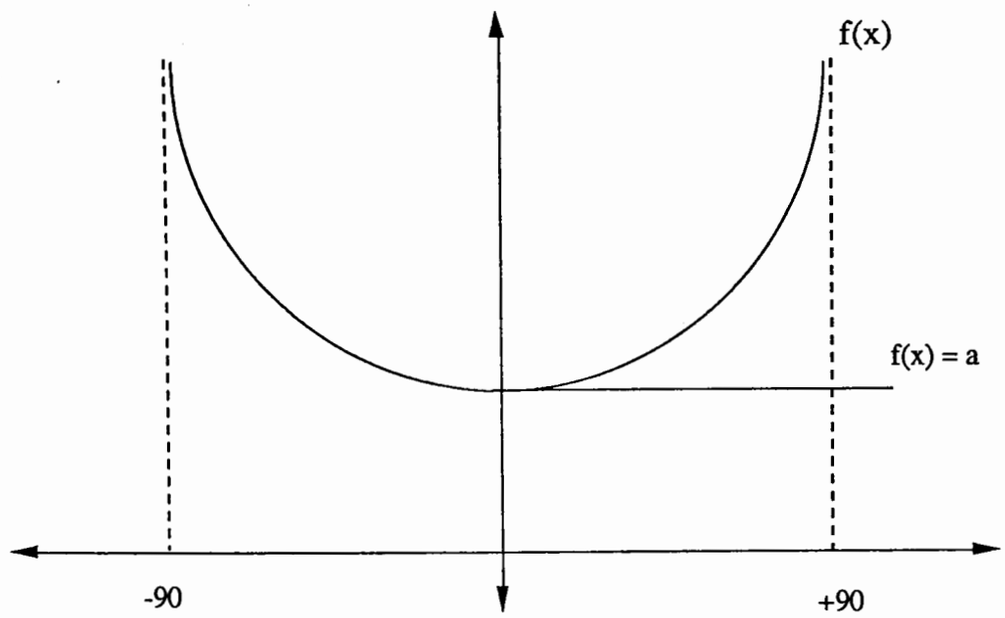


Fig 4.4 Distance to a wall as a function of angle

where a = distance to wall and x = angle

But the infra-red sensor has a maximum range of 1800mm. The longest wall that can be seen at any time (where a = 100mm) is:

$$1800 = 100 / \cos(x)$$
$$x = 86.8 \text{ degrees}$$

$$\text{Max wall} = 2 * \tan(86.8) * 100$$
$$= 3577\text{mm}$$

As the current accuracy of the infra-red scan is one observation every 1.6 degrees, observations can be recorded at 86.4 and 88 degrees. At 88 degrees the distance to the wall will be:

$$f(x) = 100 / \cos(88)$$
$$= 2865 \text{ mm}$$

As the maximum range of the sensor is 1800mm the wall at 88 degrees is out of range. The maximum angle will therefore be 86.4 degrees. At 86.4 degrees the distance to the wall is:

f(x) = 100 / cos(86.4)
= 1592 mm

Due to the logarithmic nature of the infra-red sensor the accuracy at 1800mm is very poor. It is therefore desirable to limit the angle in which to detect walls. As 100mm is the smallest distance that can be measured to a wall and it is unreasonable to need to place all points 100mm from walls a compromise must be reached. A reasonable range from a wall might be 1000mm.

At 1000mm the maximum wall angle becomes:

1800 = 1000 / cos(x)
x = 56.25 degrees.

and wall length becomes:

max wall = 2 * tan(56) * 1000
= 2965mm

4.2.4.1 Wall pattern

Given the 100mm to 1000mm constraint described above the graph becomes:

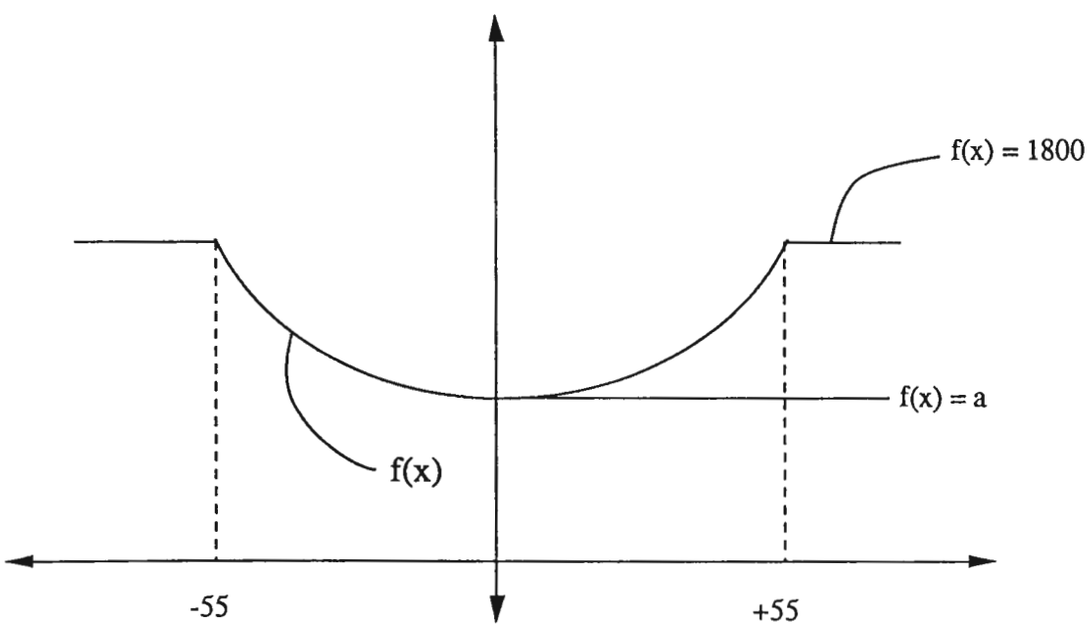


Fig 4.5 Distance as a function of angle

4.2.4.2 Detect Wall pattern

Having determined the pattern a wall returns a method of finding this pattern in a sample scan is required.

The required pattern follows

$$f(x) = a / \cos(x)$$
$$\text{where } \{ 100 < a < 1000 \}$$

This can be detected by looking for flat spots in the scan and checking that the observations on either side are increasing as the robot rotates away from right angles to the wall.

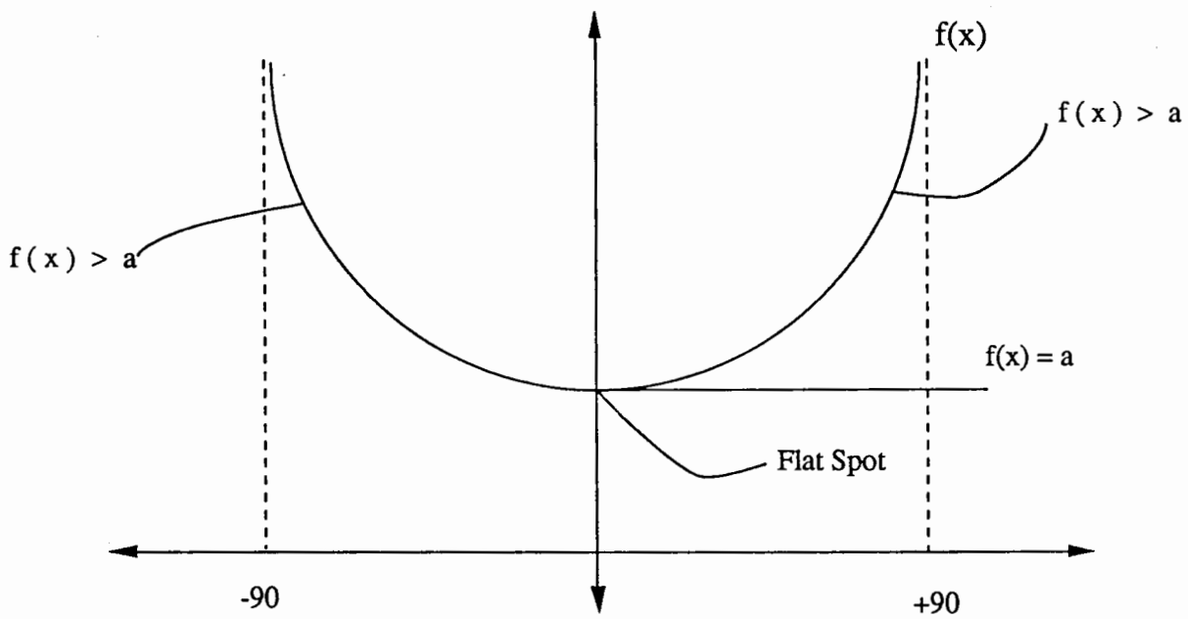


Fig 4.6 Distance to a wall as a function of angle

That is simplified by finding d f(x)

$$d\ f(x) = a \tan(x) / \cos(x)$$

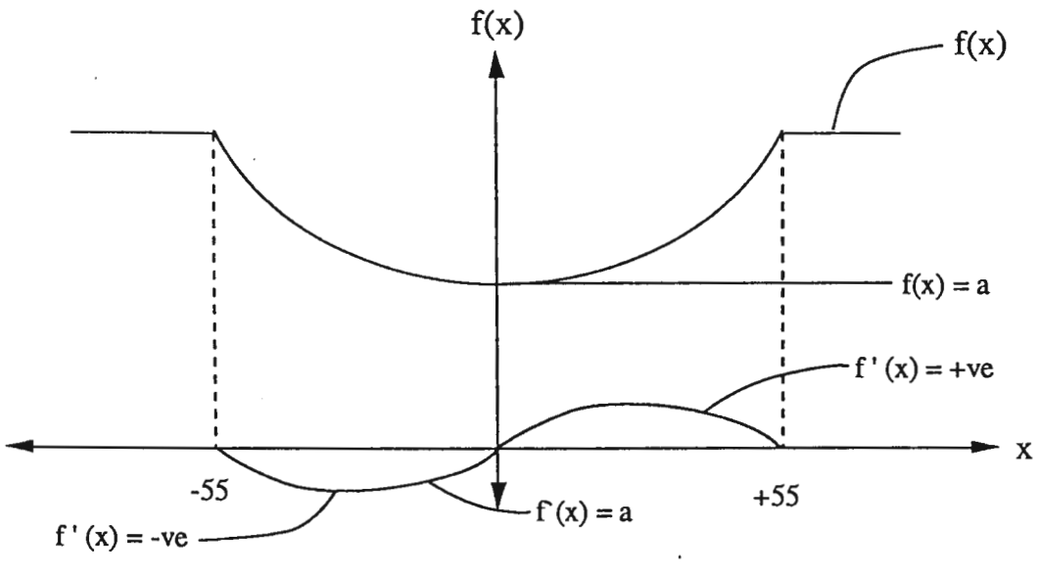


Fig 4.7 Derivative of distance to a wall

Further differentiation gives

dd f(x) = a * tan(x) ^2 / cos(x) + a / cos(x)^3

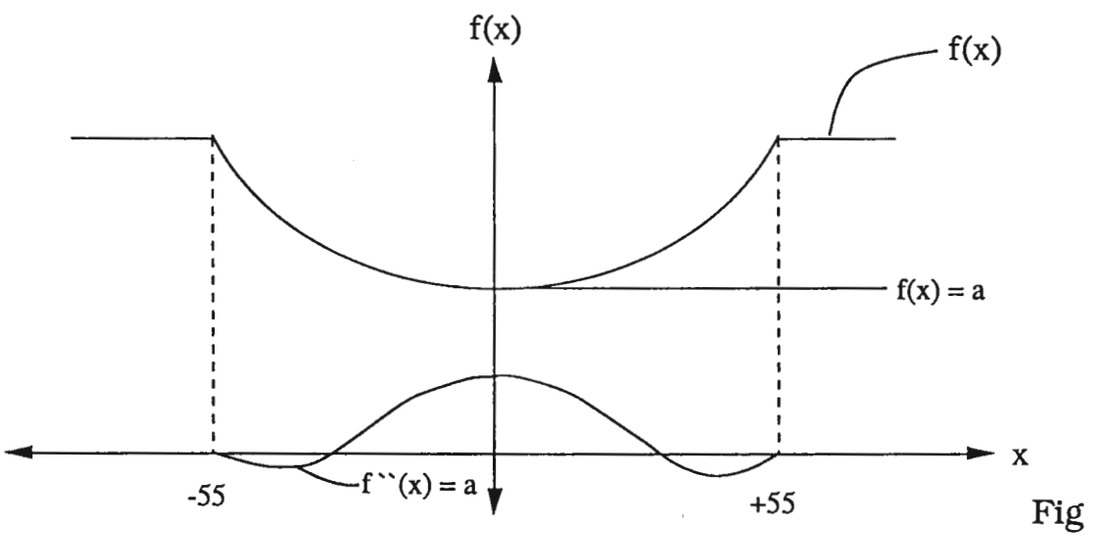


Fig 4.8 Second derivative of distance to a wall

which is a symmetric curve and simple to detect by the robot software i.e.,

f(x) = scan[x]
d f(x) = scan[x] - scan[x - 1]
dd f(x) = d f(x) - d f(x - 1)

The wall detection algorithm used is as follows:

- Search d f(x) for zero values

- At each value of x where d f(x) = 0
check that dd f(x) is symmetric about x
i.e.,

For all { a = 1 to 20 }
if dd f(x + a) = dd f(x - a) then this is a wall

In practice, walls with less than +/- 20 degrees of symmetry were accepted but were weighted as uncertain walls.

4.2.5 COMPARE WALLS

Once all walls have been found in both the current infra-red scan and the infra-red scan stored in the environment map, they must be matched and compared for changes in position.

The wall list for each scan is sorted by the distance to the wall and the wall width. The distance is significant as the infra-red sensor is more accurate at close range. The second sort criteria is the wall width. The wider the wall the higher the probability is that it is a wall.

Walls from the environment map are taken one at a time and matched with a wall from the current scan. Walls are matched by their angle from north and are considered to be matched if the angle from north to the wall in the stored environment map is within 10 degrees of the angle from north to the wall in the current scan.

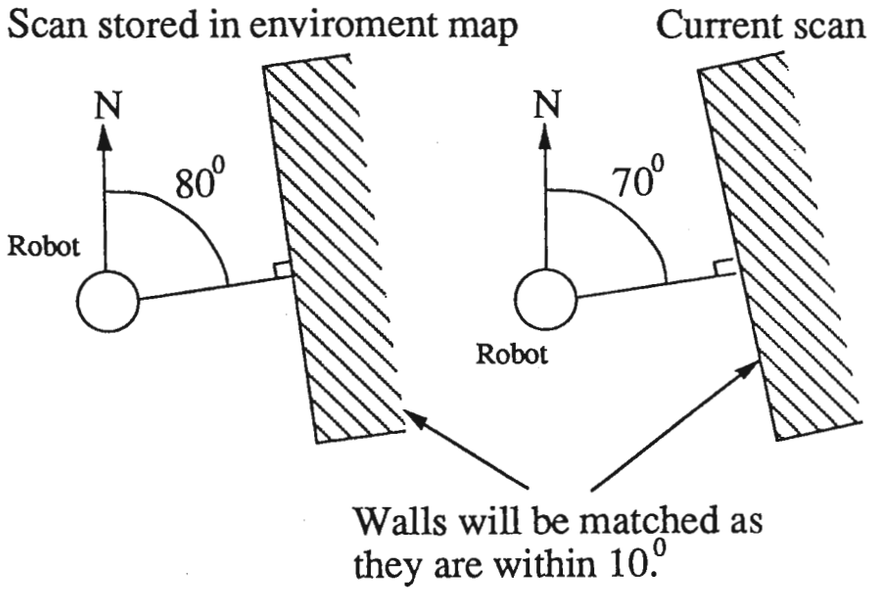


Fig 4.9 Compare two scans

Once a wall in the stored map is matched to a wall in the current map, the distances to the walls are compared. The first matching pair of walls that are displaced by more than 10mm are used in the calculation of the error vector.

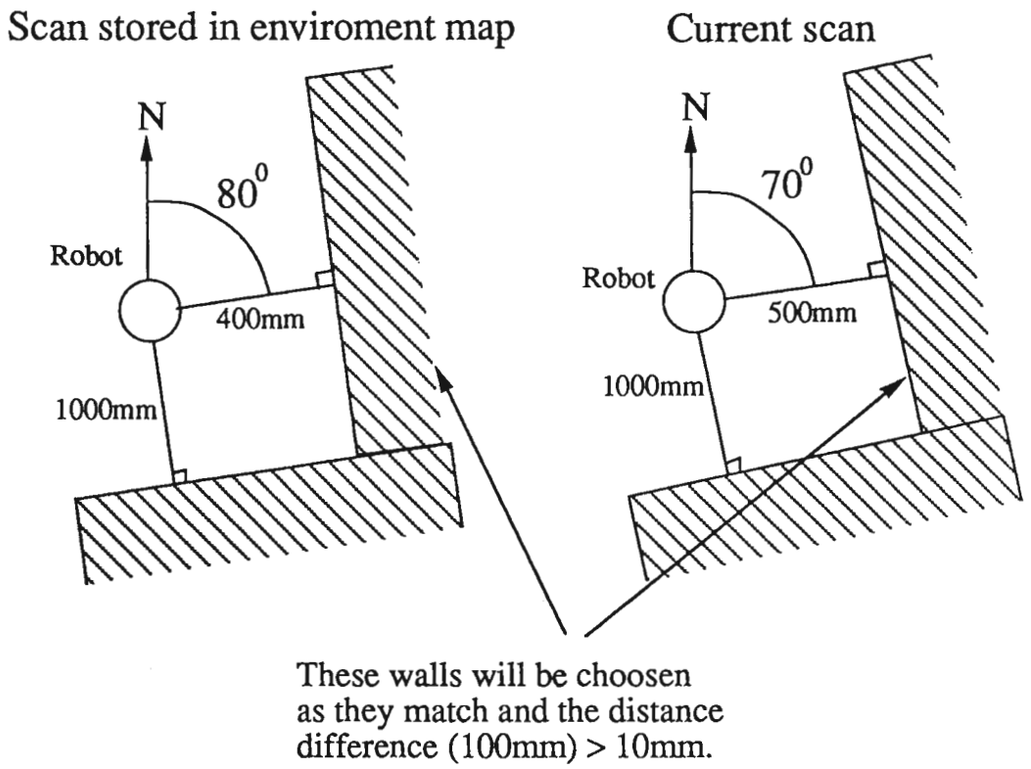


Fig 4.10 Compare position against two walls

The error vector is calculated using the first two walls selected above. If the distance to the wall in the scan in the environment map is less then the distance to the wall in the current scan then the error vector has an angle equal to the angle from north to the wall. The error vector's distance is equal to the difference of the two distances.

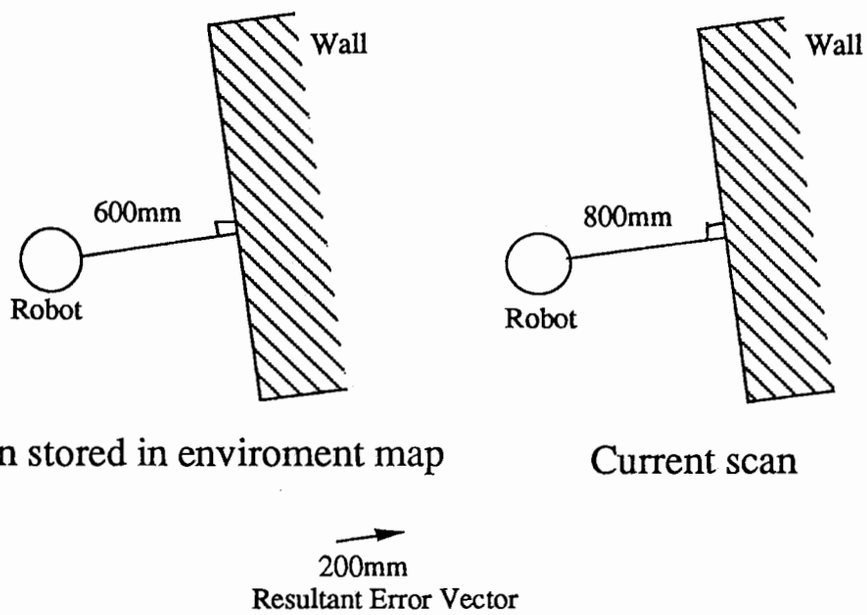


Fig 4.11 Calculate Error Vector

If the distance to the wall in the stored scan is greater then the distance to the wall in the current scan then the error vector's angle is the angle the wall is from north plus 180 degrees. The length of the error vector is once again the difference of the distances.

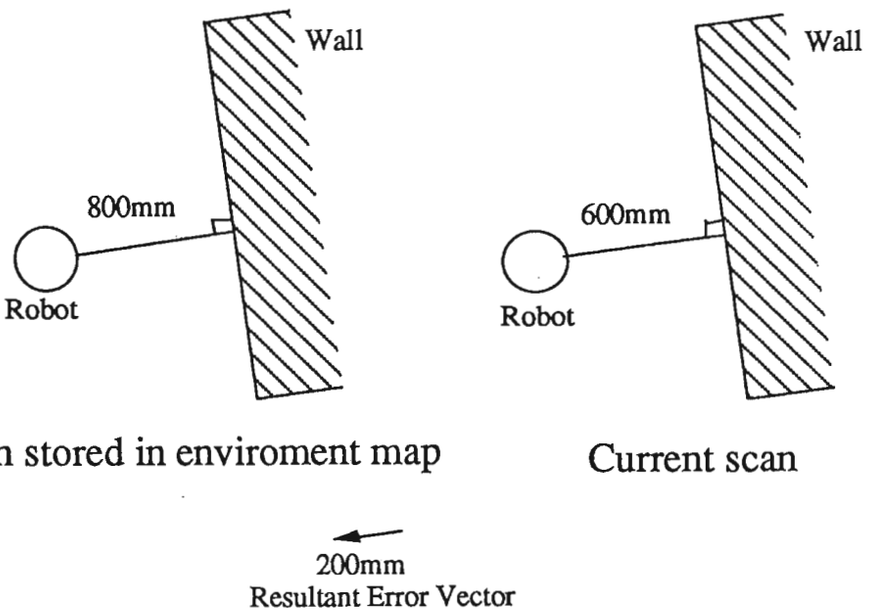


Fig 4.12 Calculate negative error vector

4.3 SENSORS

4.3.1 DIRECTION

A simple compass was selected to determine the robot's direction. It consists of a 20mm x 1mm needle mounted in a clear case filled with fine clear oil. To detect the presence of the needle a beam of light was directed through the compass. To avoid magnetic interference the light was delivered via a fibre optic tube to the top of the compass. A second tube collected the beam from underneath the case and directed it to a photo diode.

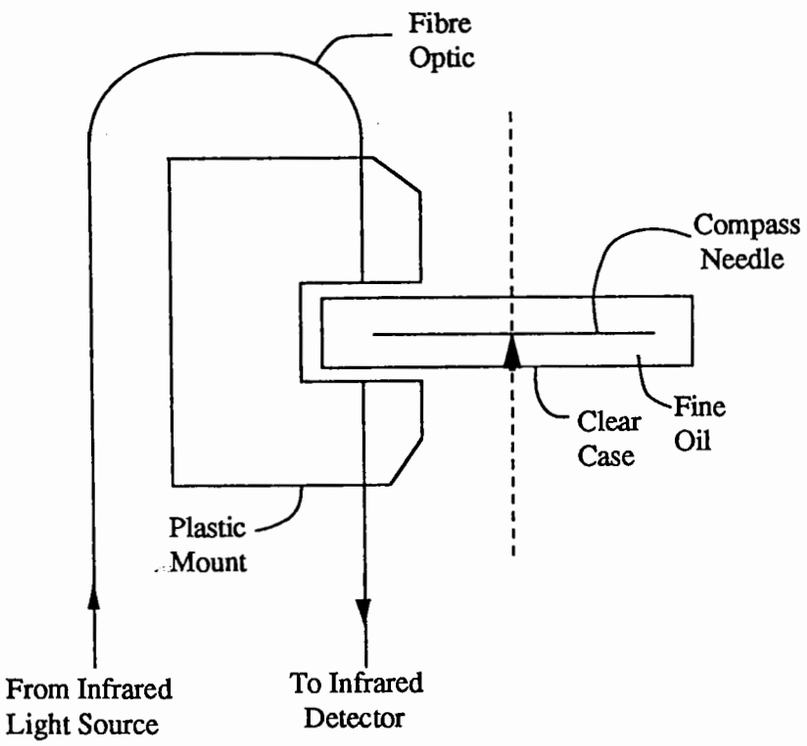


Fig 4.13 Compass detection configuration

This configuration has two major draw-backs. The first is that the needle is detected when the robot is facing both north and south. The second and more serious problem is that the robot must turn to position the needle over the optic sensor. All movements of the robot induces significant movement in the compass needle requiring a settling period. This makes it impossible to detect north or south as the robot is moving.

4.3.1.1 Deflection timing.

By placing an electromagnet near the compass it is possible to deflect the needle. By timing the travel of the compass needle from its settled position until it cuts the light beam the angle from north to the detector can be calculated.

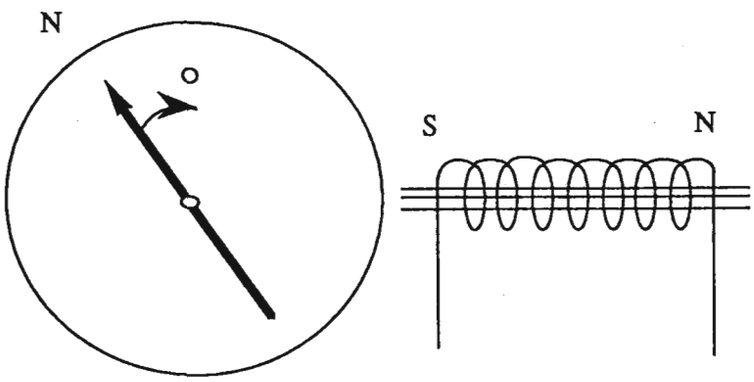


Fig 4.14 Deflect compass

This configuration overcomes the problems with needle settle time. To find north the robot waits until the compass has settled, energises the electromagnet, counts the CPU cycles until the light beam is broken and uses this value to table look up [1] the size of turn required to point the robot to north.

4.3.1.2 Detect North

The configuration used to position the needle over the sensor can also be used to tell north from south. To achieve this, the robot's compass algorithm must be expanded to include the following scenario.

When the robot is turned clockwise and the needle is detected it is guaranteed that the robot is pointing to the north east or the south west. Due to the problems with settle time it is not possible to determine by how much the needle has over-run the sensor. Over-run is guaranteed, as the compass is filled with oil which impedes the movement of the needle within the compass shell.

To use the algorithm to point the robot north the robot must first be turned to face to the north east. If the robot is facing north east, energising the electromagnet magnet will deflect the compass needle clockwise. It will cut the light beam detector and come to rest facing the electromagnet magnet.

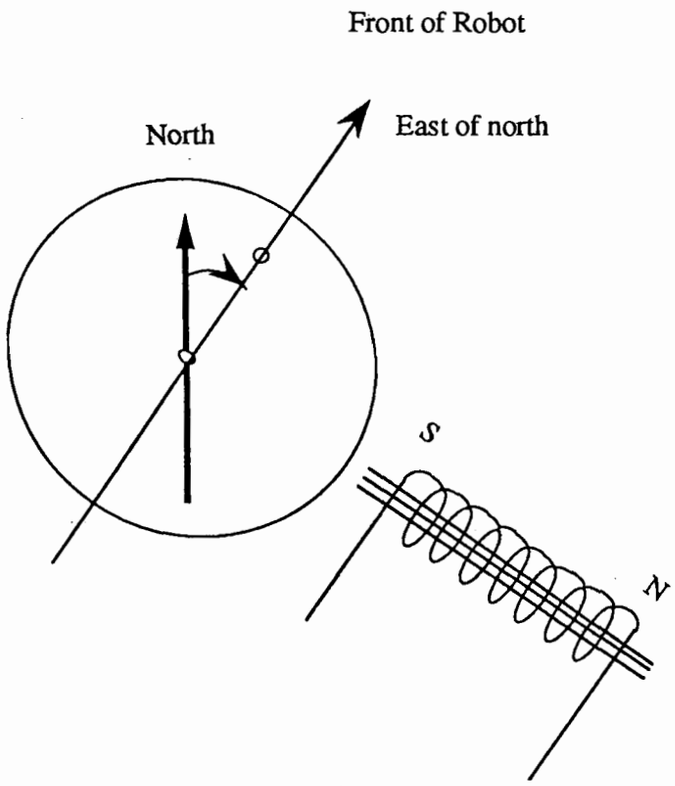


Fig 4.15 Deflect compass to North East

If the robot is facing south west when the electromagnet magnet is energised the compass needle will deflect anti-clockwise.

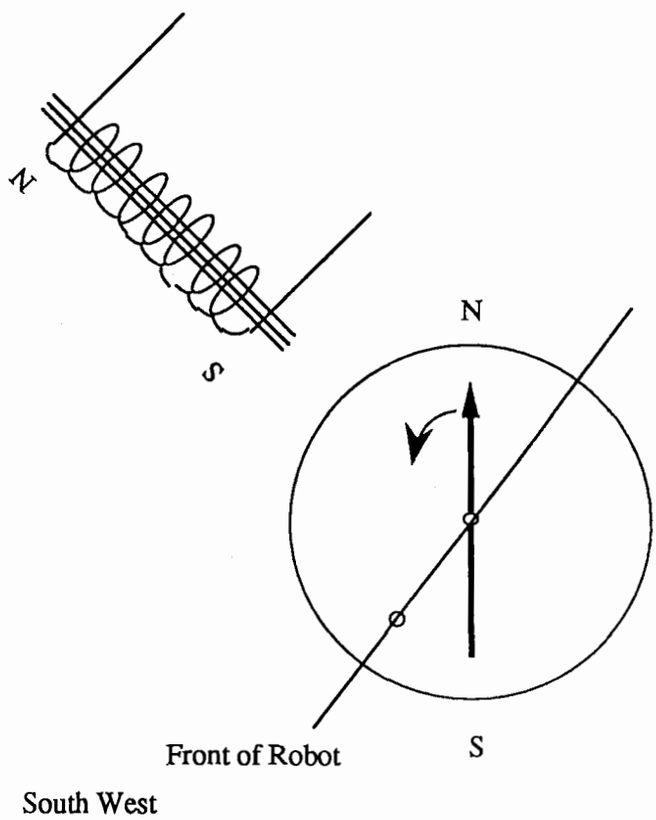


Fig 4.16 Deflect compass to North West

The needle will not cut the light beam detector but will come to rest facing the electromagnet magnet. So by limiting the cycle count for deflecting the needle it is possible to distinguish between north and south (i.e., south will time the count out). If the count does time out the robot should turn 180 degrees and apply the algorithm to face the robot north.

4.3.1.3 Compass Accuracy

The accuracy of the compass system depends on two factors. The first is the speed of the CPU used to time the compass' deflection. The second is in the use of the error information calculated from the deflection time. To use this error information the robot must be able to rotate accurately.

The wheel sensors selected on the test platform give 218 divisions per circle. This means that the maximum repeatable accuracy possible by the robot's compass system is 1.6 degrees.

The micro-controller uses a cycle time of 9 micro-seconds. A minimum loop test for needle deflection with time out check requires only 5 CPU cycles.

Label	Instruction			Cycles
	MOV	A,#0	;	Not in loop
LOOP:	DEC	A	;	1
	JB	FOUND	;	2
	JNZ	LOOP	;	2
			;	Time out

Fig 4.17 Source code of deflection timer

The above loop will time out after 0.0069 seconds. Trials with the selected compass indicated full deflection occurs after 2.7 seconds.

This indicates the accuracy of the robot's compass system in the test platform is driven by the wheel divisions not the CPU speed. The test vehicle has an accuracy range of 1.6 degrees (or+/- 0.8 degrees).

4.3.1.4 Error due to Electromagnet.

Several precautions have been taken to minimise the effects of the remaining magnetic flux in the electromagnet and the placement of metal components around the compass.

To minimise the effects of residual magnetic flux the magnet is shorted by the switching relay used to energise it. This quickly decays the field and to allow for this a delay loop is included in the control program.

The electromagnet is the only metal component mounted near the compass. Its effect is negligible for three reasons. The first is that the electromagnet is permanently fixed in one position on the compass chassis. The magnet is placed at 90 degrees to the needle deflection sensor. This means that although the earths magnetic field is distorted by its presence, it is a constant distortion. As only north is read from the compass and all other directions are deduced using the wheel sensors, the effect of the electromagnet in a de-energised state is negligible.

4.3.1.5 Current Heading

One of the functions of the robot's micro-processor is to maintain a register of the current bearing. It does this by incrementing the register each time a wheel division is detected when the robot is turning clockwise. Likewise the count is decremented for each division when turning anti-clockwise. The counter is also maintained while in forward motion to make allowance for one wheel moving faster than the other.

Due to the problems with the compass needle settle time it is not possible to reset the count to north whenever the needle is detected. Instead the count must only be reset to north when the north search algorithm is executed via a north command request.

The direction register is used by the robot controller to accomplish several tasks. One of its uses is in calculating the turn size and direction to turn the robot near to north before applying the north search algorithm. This enables the robot to turn to the north east sector through the least number of degrees. The standard north search algorithm could turn through 270 degrees before finding north (south east sector start is the worst case).

Other uses the robot controller has for the bearing register is to assist in the storage of the scan results, direction statement on the status line and calculating the smallest move in the turn to heading command.

4.3.2 DISTANCE MEASUREMENT

Like the compass needle sensor, light sensors were used for the measurement of distance travelled by the robot. This time the sensor used a series of reflective stripes on the side of the wheels to reflect the light. The robot has a stripe every 4.5mm of forward travel.

In movement commands issued by the Navigator or Map Edit programs the distance to move in wheel divisions are included. When the robot controller interprets such a command it copies the distance into two wheel count registers. The controller then energises the wheel motors and monitors the sensors. As each strip on the wheels is detected it decrements the count register for that wheel.

At the end of each polling loop the wheel count registers are compared. If the difference between the counts is greater than two the wheel that is ahead (lesser count) is slowed until the other wheel catches up.

When the count register's values drop below ten, the motors are slowed by pulsing their supplies. This enables the robot to stop smoothly. In the slow mode a counter is set running for each wheel. If the count exceeds a "Dead" count then that wheel must have stalled so full power is applied to that motor until the next strip is detected.

When either count reaches zero, the current to both motors is cut and the brake solenoids are energised until the robot is halted.

4.3.3 VIEW

To support the wall detection and point comparison process a sensor that is capable of measuring the distance from the robot to objects was developed[8]. The sensor measures the intensity of a beam of infra-red light, reflected from the object. Several problems are apparent with sensors that use reflected light for measurement.

The first problem is the potential effects of external light sources on the sensor. Light from external sources can enter the sensor and interfere with the result by either overloading the sensor or adding to the intensity of the reflected beam. Steps were taken to minimise these effects in the sensor used on the test vehicle.

The second problem is related to the ability of the target object to reflect light. Shiny objects reflect more light and so appear closer than they actually are. The design also attempted to overcome this problem but test results indicate little improvement.

The final problem encountered is due to the relationship between reflected light and its intensity. This relationship makes it difficult to build an interface that has large enough signal range to accurately measure the intensity at larger distances.

The design used consisted of a single modulated light source and two detecting points. The results of the infra-red light detectors are subtracted to produce an output signal with the DC component of the signal rejected. The configuration is as follows:

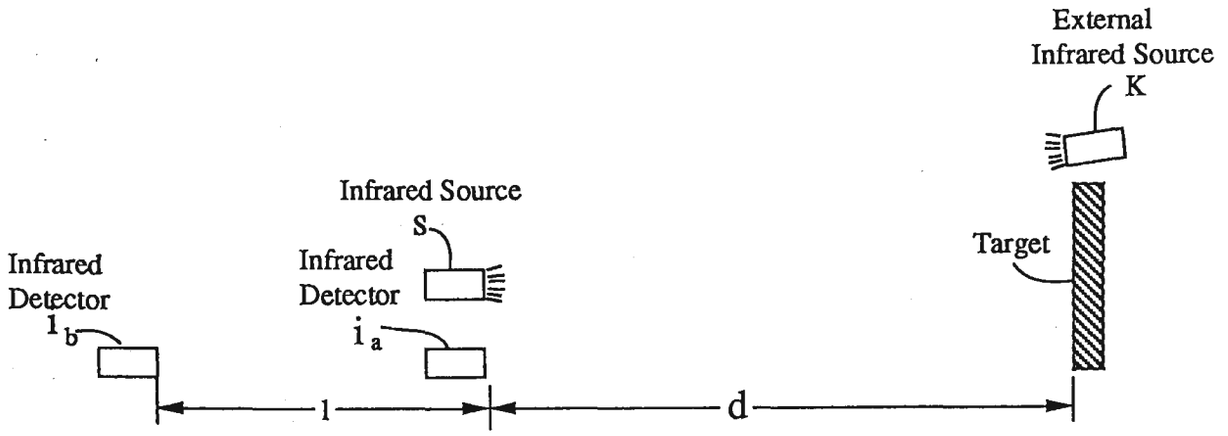


Fig 4.18 Distance measurement configuration

The intensity at point a follows the function

$$i_a \simeq \frac{s + K}{d^2} \quad \text{When S is on}$$

$$i_a \simeq \frac{K}{d^2} \quad \text{When S is off}$$

where s is the intensity of infra-red light at the target due to the source S and K is the infra-red intensity at the target due to external sources. Likewise the intensity at point b follows the function

$$i_b \simeq \frac{s + K}{(d + l)^2} \quad \text{When S is on}$$

$$i_b \simeq \frac{K}{(d + l)^2} \quad \text{When S is off}$$

The configuration subtracts the intensity at b from the intensity at a to give:

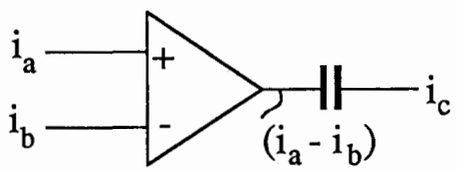
$$i_a - i_b \simeq \frac{s + k}{d^2} - \frac{s + k}{(d + 1)^2}$$

When S is on

$$i_a - i_b \simeq \frac{K}{d^2} - \frac{K}{(d + 1)^2}$$

When S is off

The DC component of the signal is then removed to give:



$$i_a - i_b \simeq |(i_a - i_b)|_{on} - |(i_a - i_b)|_{off}$$

$$i_c \simeq \frac{s}{d^2} - \frac{s}{(d+1)^2}$$

The function that represents signal value vs distance is a function that is similar to an inverse square and suffers from sensor overload at close ranges as shown below. Appendix A.5 gives an example of data collected from the sensor on the test vehicle.

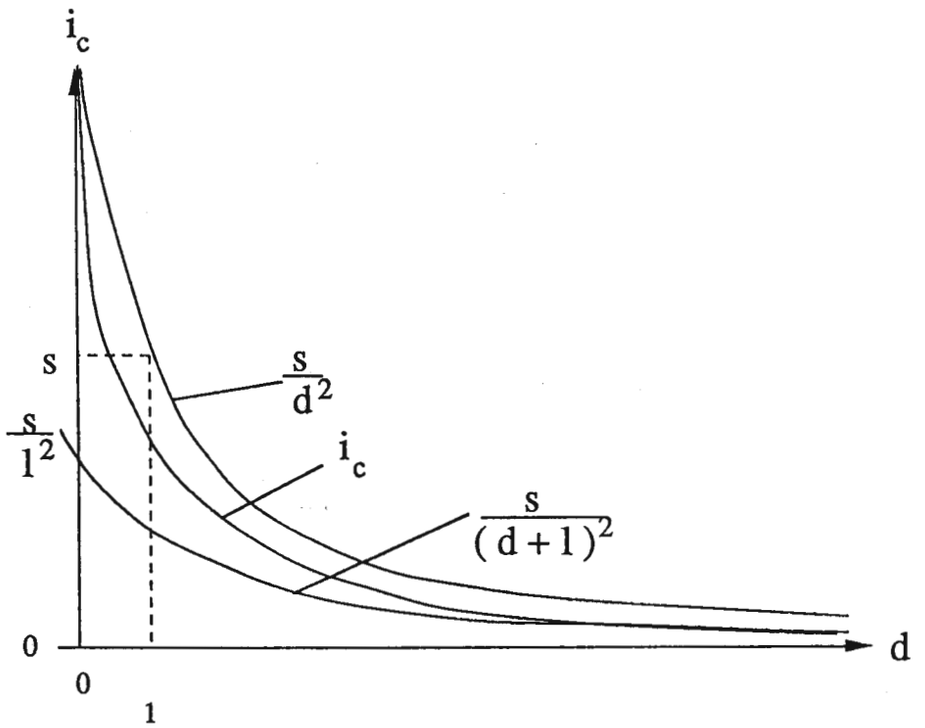


Fig 4.19 Intensity as a function of distance

The result at c is free of the external light source. In practice infra-red light from external sources did effect the resultant signal as most external sources are also modulated. Filtering was required to improve the sensor's ability to reject infra-red light from external sources.

4.3.3.1 Infra-red Sensor Interface

The light source used consisted of a string of infra-red LEDs. To avoid interference from static infra-red sources (like the sun) the LEDs were modulated. This also reduces the effects of fluorescence light. The LEDs were driven by a power amplifier driven from an oscillator running at 10Khz. This frequency was sufficiently high to avoid interference yet low enough to permit the use of cheap operational amplifiers.

The detectors were amplified before filtering. Simple first order filters were used before the signals were subtracted. Further stages of amplification and filtering were then applied to the signal. A precision rectifier was used and the signal scaled for input to the Analog to Digital converter. This gave an opportunity to scale the signal to full

range on the Analog to Digital converter, in an attempt to squeeze all the precision possible from the measurement device.

This configuration when fully adjusted had a maximum range of 1600mm and a minimum distance of 100mm.

Conversion to a digital value was via a National Semiconductor Analog to Digital converter. This device is capable of a conversion every 100 micro-seconds enabling some 40,000 observations per 4 second scan circle. Only 218 x 4 observations were used as this was found to give the most reliable result without tying up the micro-controller recording the results.

4.3.3.2 Scan Recording.

To simplify the hardware required to keep track of the sensor outputs they were polled sequentially. This proved to be a suitable alternative to hardware driven interrupts. The polling loop tests each sensor of the robot in turn. If a change in status of a sensor is detected, the polling routine sets a flag to indicate an event and returns to the current move command control routine. The polling loop is executed as often as possible and was found to be reliable, not missing any events during the vehicle's testing period.

One of the events the polling loop checked for was the infra-red sensor returning a distance below a preset (in the test case 100mm). This use of the infra-red sensor formed part of the bump detection and avoidance system.

A second element of the polling loop was the recording of infra-red sensor results during the scan command. This was achieved by using a toggle that activated the storage routine. When the polling loop detected that an infra-red sensor reading was available, the toggle was checked and if set the compass register is accessed and the result used as an index by which to store the infra-red sensors value.

The use of the toggle proved a solution to power supply problems evident immediately after the drive motors were energised. The drive

motors induced a spike that propagated through the infra-red sensor's operational amplifier stages to give false readings. This problem was solved by turning the robot through 10 degrees before toggling the record flag to store infra-red sensor values.

4.3.3.3 Scan Control Procedure

Once the scan command has been received by the robots micro-controller it issues a series of calls to the move procedure. The sequence is as follows.

1 - Turn to the north east
2 - Find north
  a - Deflect Compass
  b.- Wait for Needle Crossing
  c.- Turn amount given by Deflection table lookup.
  d.- Repeat until deflection = 0
3 - Turn 10 degrees Anti-Clockwise
4 - Turn 380 degrees Clockwise
  a.- At zero degrees (north) start infra-red recording
  b.- At 360 degrees (north 2nd time) stop recording
5 - Find north
  a.- Deflect Compass
  b.- Wait for Needle Crossing
  c.- Turn amount given by Deflection table lookup.
  d.- Repeat until deflection = 0

Steps 1,2.a, 2.b, 2.c & 2.d are necessary to provide a reference point which will be used to compare scans. Step 3 is necessary to provide a run up so that the robots power supply will be totally stable and the scan will be done at a constant speed. Step 4 gives enough overlap of a complete circle to guarantee 218 observations. Step 5 provides rotational accuracy for the next path of the selected route.

4.3.3.4 Scan Averaging

Further improvements in the results of the scan command were achieved by averaging the results. The simplest form of this was implemented in the recording routine by averaging each angle over four observations. This was done by keeping a count of the number of observations accumulated in the storage table. When the count reached 4 no further observations were taken until the robot turned to the next tick.

It was found that even at full speed four observations were possible before the next tick was reached. In fact 64 observations were possible but 4 provided a slice that contained more crisp detail of the local environment.

4.3.3.5 Scan Smoothing

The averaging of each observation in a scan over 4 readings produced promising results. These could be further improved by smoothing using an algorithm that took account of adjacent observations.

Two smoothing algorithms were chosen that were applied alternatively to the results. Four applications of the two smoothing algorithms seemed to produce optimum results. Further iterations suffered too much from the effects of the logarithmic nature of the infra-red sensor readings.

The first smoothing algorithm used the following formulas

$$x(i) = (x(i-1) + 2*x(i) + x(i+1)) / 4$$

The second formula used the same weighting but over a wider space and used the formula

$$x(i) = (x(i-2) + 2 * x(i) + x(i+2)) / 4$$

The algorithms were written in 8051 assembler and ran very quickly.

4.3.3.6 Data Upload

The transfer of scan results from the robot's micro-controller to the PC required software at both ends. The transfer was achieved over a serial RS232 line. The line had a maximum speed of 9600 baud but the PC is unable to process the information at this speed for any sustained period. The use of XOn/XOff protocol did not improve the ability to handle this speed. Several steps had to be taken to resolve the problems.

The most obvious solution was to minimise the amount of data transferred. This was done by blocking the data into records of five observations. The scan angle was removed and all 218 observations were always transferred. This resulted in a constant block size of 44 lines of 5 integer observations. Each integer digit required 3 bytes and was separated by commas with line terminating line feed characters giving a total of 880 byte blocks.

It was found that only one line of the block could be processed by the receiving PC at a time or overrun would occur. This problem was solved by flagging XOff to the micro controller after each line. When the line was stored by the PC XOn was sent and transmission continued. This protocol proved to be very reliable.

4.3.3.7 PC Software

The upload software in the PC also handled the conversion of the scan observations from millivolts to length. This was done via a lookup table. The infra-red sensor's value (0-256) is used as an index into the conversion table and the length in millimetres read directly. This arrangement operates very fast with no detectable difference between storing direct millivolts values and storing converted millimetres values.

This conversion was initially attempted by matching a function to the millivolts vs millimetres curve. The equation was a 3rd degree quadratic which proved difficult to solve for values of f(mv). The use of approximate methods to solve the equation ran quickly on an individual observation but was unreasonably slow when used on a complete scan (218 observations).

To afford the maximum flexibility a program was written that could create the conversion table and store the results in a file. The program when used in conjunction with the robot, moved the sensor from a fixed wall in 100mm steps. Observations were taken at each step and the smoothing algorithm was applied until all 256 mv entries were extrapolated.

The major advantage of using a conversion table stored on disc file (other than conversion speed) was the ability to re-calibrate quickly. This meant that after modifications to the infra-red sensor, a new table could be generated in minutes and the navigation and edit programs used without the need to recompile or relink. The procedure that controlled the upload of data minimised the access to the conversion table file only accessing the file upon the first conversion and storing the table in common memory for access in future data uploads.

4.4 MAP STORAGE

4.4.1 Access Path

4.4.1.1 Edit

The edit program presents the operator with rooms containing points. The room concept was used to aid with the design of environment maps. The operator selects the required points and labels them first by room then by individual point.

The edit entry screen forces the operator to create a room, then enter the points in that room. The map can be examined room by room, each room screen listing the points within that room.

The storage structure supports this function through the room list. The program can quickly locate a room on the list. Once the room is selected, the room's point list lists all the points in that room. This supplies the display function with a list of points by making them directly accessible from the room.

Paths are created by selecting the start point, choosing the room (which gives a list of the available points) and selecting the end point for that path. The path record is created simply by adding the path to the path list for the point already chosen. The edit program must operate this way or a search for the start point would have to be executed. As no point list exists that contains all the points the search would be more indirect.

4.4.1.2 Navigation

The algorithm chosen to search for the shortest path was a simple depth first graph search. This type of search examines every path from a point to all its adjacent points (except the one it just came from).

The chosen structure supports this search very well by supplying a list of available paths attached to each point. The algorithm simply

starts at the source point and searches the path list of that point visiting all the point's neighbours by moving down the path list.

The search algorithm keeps track of the cost (route length) of the current route. It does this by adding the distance to a total as it moves to a point's neighbour and subtracting as it retreats. Once again this requirement is well supported by the storage structure chosen as the distance is stored on the path record.

It was necessary to slightly modify the original structure to support the search as a visit flag was required so the search will not loop back upon itself. This was achieved by adding a flag to the point record.

Source and target points were chosen by the navigation program using the room list structures. The operator is presented with a room with its points displayed. This was achieved using the point list for the first room. If the required point was not within the selected room the next room is selected from the room list. Once the point is found the memory address of the point is recorded for reference by the search algorithm.

4.4.2 Storage structures

Three simple structures were used to store the environment maps in memory for both the edit and navigation phases. The structures were a room list, a point list for each room and a path list for each point. The structures operated via the use and storage of pointer fields. The pointer fields return the address in memory of either further elements in the list or to lists of other structure types.

4.4.2.1 Rooms

The room list was the anchor point of an environment map. All searches for points started at the room list. Each room has a point list

connected to it. In this way a point search algorithm can find all the points belonging to an environment map.

The room record is very simple consisting of three fields, the room name, a pointer to further rooms in the map and a pointer to the point list for that room.

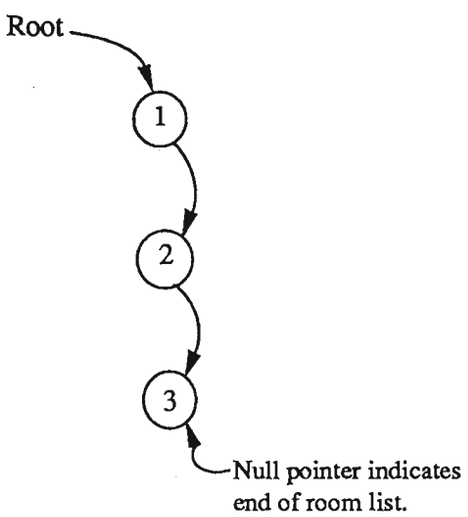


Fig 4.20 Room linked list example

4.4.2.2 Points

The point list contains information about known points for each room. This information is used to check the actual positioning of the robot against the desired position.

The point record consists of the scan results, a pointer to other points in the same room, a pointer to a list of paths leading from this point, a visited flag and the point and room name of the point. The visited flag is used by the navigation search algorithm to avoid looping.

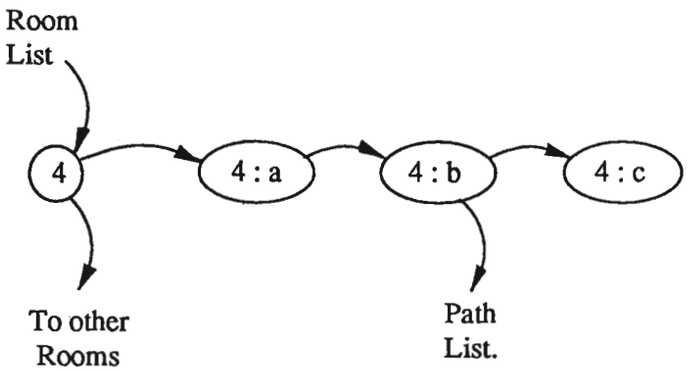


Fig 4.21 Room to synchronization points list example

4.3.2.3 Paths

The path list can contain zero or more path records. The path record contains information about how the robot can move from the current point to the point whose memory address is stored in the path record. The stored path information consists of the bearing and the length of the path.

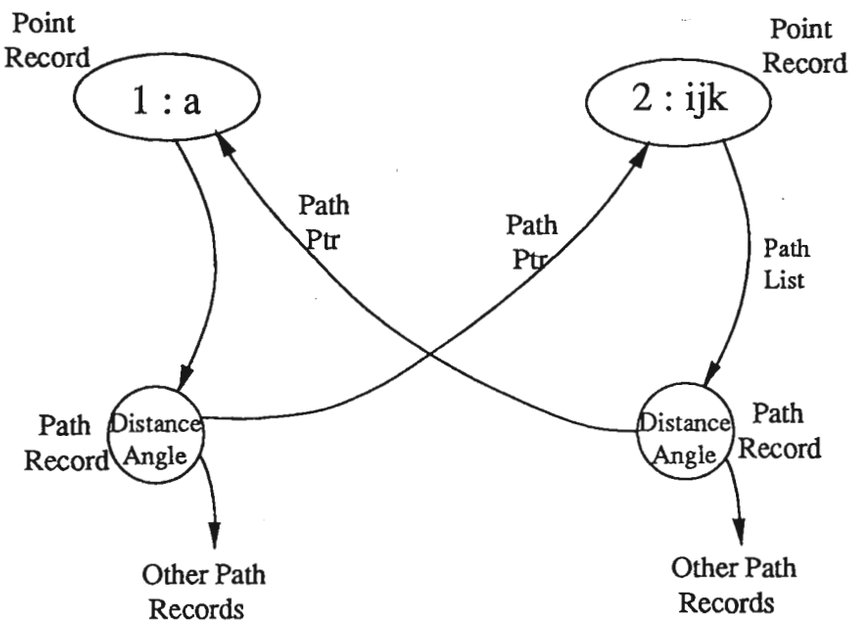


Fig 4.22 Two way links between points

The path list enables the navigation program to search for the route from the current point to the target. Once the route has been found the information is used to construct the command sequence necessary to instruct the robot to move between the points.

4.4.3 Permanent Storage

Both the Edit and Navigation phases of the system operate with the Environment map in memory. This ensures high speed operation but some method of switching between modes and other environment maps must be provided. This was achieved using DOS files[9].

Environment maps are stored in a compressed form on disk. The compression is possible as information from the room records is duplicated on the point records and need not be stored. The files are stored in binary format which uses as little space as possible.

Due to the duplicated information, two record formats only are required to store the memory environment maps. The records are similar in format to the point and path records. The permanent record used to store points on file has no path list pointer, instead this information is constructed as the map is read back. The path permanent storage record does not store pointers to points but instead stores the name (room and point) of the point it is leading to.

To simplify the load subroutine a record identifier is prefixed to each record. As there are only two record types the identifiers used are 'a' for the point permanent record and 'b' for the path permanent record.

4.4.3.1 Writing Permanent Map Files

Map files are written to disk only by the Edit program. It is not necessary to write map files from the Navigation program as the memory map is not updated as the robot navigates to a destination and exploring is not permitted.

The Environment map write to disk file routine can be broken into two parts. The first involves writing a copy of all point records to the file. The second half is to write all path records.

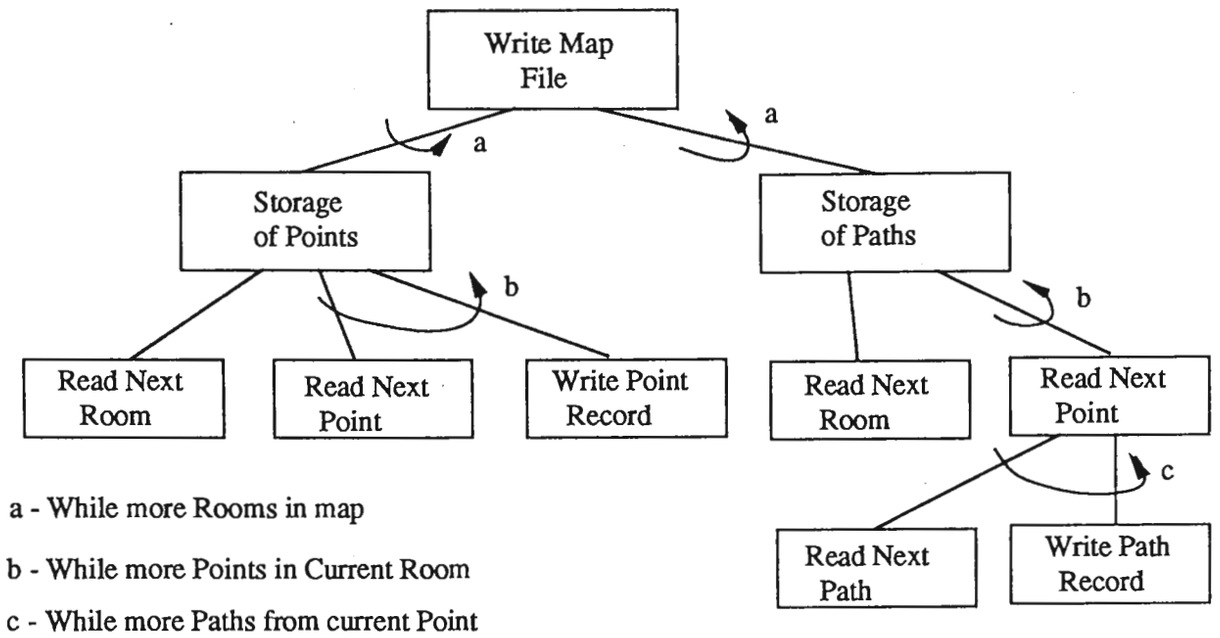


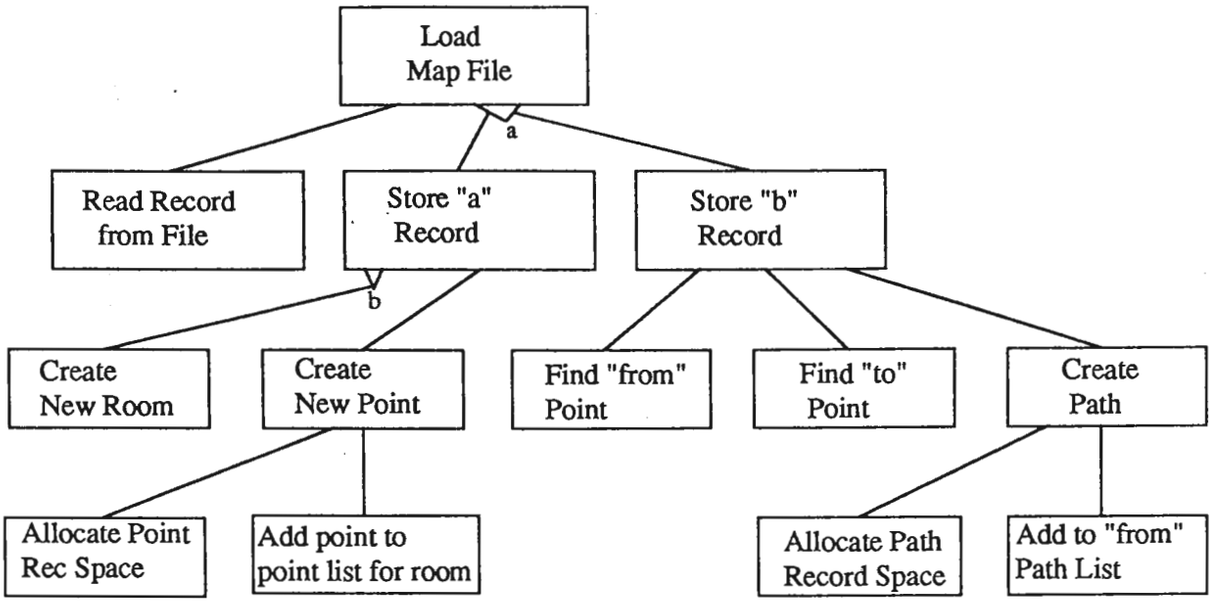
Fig 4.23 Subroutine for storage of maps in files

The storage of point records involves searching the room list to find all point lists. For each room the point list is accessed and a record is created for each point.

Once all the point records are written, all the paths must be stored. This is done by once more reading the room list to find all the point lists. Then for every point, scanning its path list and for each path that is found a path (type 'b') permanent record is written to the file.

4.4.3.2 Reading permanent record files

Environment map files are read in both the Edit and Navigate phases of the system. The process of reading from permanent disk environment map files can also be broken into two phases. The first to deal with the 'a' records and the second the 'b' records.



a - Record type, b - Does room currently Exist

Fig 4.24 Subroutine for reading a map file

As each record is read the record type is examined and the appropriate procedure is called to allocate space, store the record and create the required record links.

When a point permanent record (type 'a') is read it is first stored as a point record. The room list is then searched to see if the room name on the point record exists. If it is not found a room record is also created to add to the room list. The point must then be added to the point list for that room. This is done by adding the point to the end of the list for an existing room or as the only entry for a newly created room.

Path records (type 'b') are stored by first allocating a path record in memory. The room and point lists are then searched to find the "from" and "to" points. The path record is then added to the end of the path list for the "from" point. The memory address of the "to" point is then stored in the path record together with the distance and angle information.

4.5 Edit Support

A sophisticated map edit program was written to facilitate the creation of maps for use in the Navigation program. The editor integrates all the tasks necessary to create a map. The tasks included are reading and writing maps to disk storage, creation of rooms, points and paths, the scanning of a map and a screen display of a point's stored Scan. The map editor was written before the Navigation program.

4.5.1 Environment Map File Operations.

The map storage/retrieval part of the Edit program provides all the functions necessary to maintain several environment maps on a disk. All the commands necessary to support directory structures for storage and retrieval of map files are available without the need to exit to DOS.

The map storage/retrieval function's screen is broken into two partitions. The top lists all the map files and the lower sections lists sub-directories that the user can switch to. The operator is able to move the highlight cursor about the screen (from file to file, sub-directory to sub-directory and between partitions) to perform operations on the entities in that section of the screen.

This screen is maintained by reading the DOS directory and displaying on the top part of the screen all files ending '.pts'. All directory entries that are themselves directories are displayed on the lower part of the screen.

4.5.1.1 Sub-Directory Operations.

The operations that are available in the lower part of the screen include change directory, create sub-directories and delete sub-directories. Change directory is achieved via two function keys. The

first selects a sub-directory and makes it the current directory. The second function key switches the parent to be the current directory.

4.4.1.2 Read/Write Maps to Disk.

The operations that are available in the upper section of the permanent storage control screen include load, store, create new and delete environment map files.

The load operation is performed by highlighting a file using the cursor control keys and pressing the load function key. The program reads the highlighted file, allocates memory and stores the environment map in memory. The procedure described in Permanent Storage is used to load the selected file.

The store operation works two ways. The first is to select an existing file in which to store the environment map. This is done using the cursor control keys to highlight the desired file name and pressing the store function key. The second way is to move to the empty cell on the file part of the screen, type a new file name and press the store function key. The environment map is read from memory and stored on the permanent disk file using the algorithm described in Permanent Storage.

4.5.2 Create rooms, points and paths

This function allows the creation of Room records. This is achieved by allocating memory for a room record, storing the room name and including the new room in the room list for the current map. Rooms may also be deleted (providing they contain no points) by removing them from the list of rooms in a map. The memory is then returned to the system.

Once the required room has been created its associated points may be created. This is done by allocating memory for a point record, storing the point name and room name (as a cross check) in the

record. An empty path list is attached to the new point record. A point can be deleted by removing it from its room's point list and removing all the entries in its path list. The path list of the point to be deleted is searched and every point it is connected to has the back link deleted before the forward link is broken.

Paths are added to the map by selecting two points and creating forward and backwards links. The distance between the points is copied into both path records. The angle is stored in one and the angle plus 180 degrees is stored in the backwards path. When a path is broken both the forward and backward links must be removed.

Memory for all storage records is allocated via the operating system. As records are deleted the memory space is returned to the operating system. This arrangement allows the maximum number of rooms, points and paths to be maintained.

4.5.3 Local Point Maintenance

This part of the edit program supports the storage of scan results and the testing of environment maps. This is achieved by connecting the robot to the PC and issuing commands to the robot and recording the results.

To test a map the robot is placed at a test site and that point is selected on the map. The adjacent points are then displayed on the screen. The operator can select and perform one of the following. The angles and distances of the path to the selected point can be tested by pressing the move function key. This key sends the bearing command to the robot followed by the move command. If the robot arrives at the location described by the selected point the path is correct. If not, the angle and/or distance may be modified using the distance and angle function keys.

The other operation possible in this screen is the scan and storage of scan results for the current point. When the scan function key is

pressed the robot is instructed to perform a scan. The scan results are then read into the PC and stored in the current points storage record.

4.5.4 Hard Copy

A simple report has been created that lists the map currently in memory. This is achieved by searching the room list. The room name is printed as a heading. For each room its point list is searched. The point name is printed as a sub-heading. Each point's path list is then searched and the distance, angle and end point name are printed.

Room	Point	Target	Dist	Angle
1	ABC	1:STR	5	90
	STR	1:ABC	5	270
		2:K	6	90
		1:END	20	186
1	END	1:STR	20	286
2	K	1:STR	6	270

Fig 4.25 Sample map report

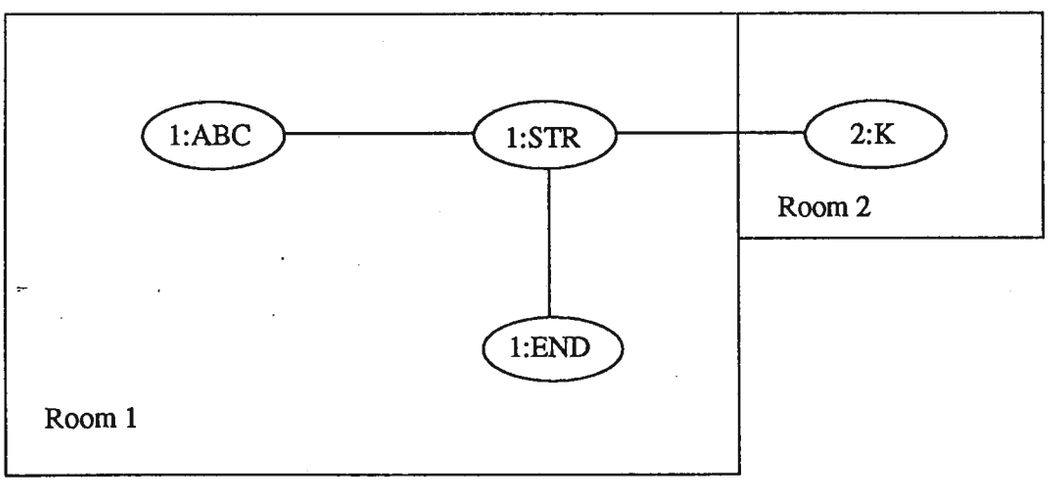


Fig 4.26 Sample map as described in listing in Fig 4.25

4.5.5 Scan Screen Display

To ensure compatibility with all IBM PCs and compatibles, advanced graphics were avoided. Instead character graphics were used on an 80x25 character screen[9]. Two views of a scan were provided.

The first and simpler display was to use the 80 columns wide to represent a portion of the 218 scan observations. The 25 lines represented the distance to objects within view of the optic sensor. The largest scan value for the point being displayed was found and then scaled to 25. This process maintained maximum definition for each point displayed. The scroll keys were used to display the next 80 observations of the scan.

The second display uses the middle of the screen line 25 as the centre of the scan. The scan is then drawn around that point to give a radar type view of the local environment. The scroll keys once more allow the part of the scan being displayed to be changed.

This part of the edit program, although not useful in testing the accuracy of maps, provided a means to visualise the scan. This display was useful in developing the concepts of the feature analysis routine used in the navigation position correction routine.

5. Test Results

5.1 Summary

Five simple tests were performed to measure the success of the robot. The tests were designed to aid in the development of its dead reckoning and position correction functions and to demonstrate its ability to navigate from point to point.

5.1.1 Mechanical accuracy

The first group of tests demonstrate the mechanical properties of the platform developed.

A test map consisting of two points (1A and 1B) connected by a single path 600mm long, 300mm from a wall was created (Fig 5.1).

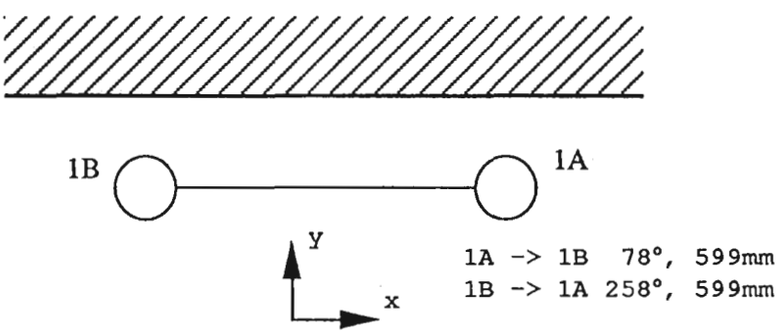


Fig 5.1 First test map layout

This was done by coding the map using the map editor and placing the robot at each of the points to record the scan patterns. The path vector was then fine tuned using the map editor to move the robot from point 1A to 1B. The distance to the wall was checked and the path vector modified. This process was then repeated and the floor marker for 1B moved to mark the resting position of the robot. The robot was once again placed at point 1A and instructed to move to 1B. The robot was able to repeat the move coming to rest on top of the marker for 1B. The results of this test (table 5.1) showed that a simple

path leg could be reliable repeated with the robot able to accurately control its angle and distance of travel.

Test	Point	Length Error	Error to Wall
1	1B	6mm	9mm
2	1B	15mm	15mm
3	1B	3mm	0mm

Table 5.1 Error Size for a Single Leg Journey

The return journey (i.e., 1B to 1A) was not as reliable (see table 5.2). This is due to 2 factors. The first is error introduced into the robot's position as it turns. This error is caused by unequal wheel speeds. The controller's software is able to keep the wheels within two wheel divisions of each other but even this small discrepancy causes the robot to spiral off the spot. The second problem is due to the number of divisions per circle not being exactly 218. This means that 180 degrees is not 109 divisions and so the robot is unable to exactly face the direction from which it came.

Test	Point	Length Error	Error to Wall
1	1A	10mm	-94mm
2	1A	-13mm	-105mm
3	1A	20mm	-155mm

Table 5.2 Error Size of Returned

5.1.2 Error Correction in one direction.

The next test was used primarily as a tool to develop the position correction functionality. In this test error was considered in one direction only. This was done by removing objects from sensor range at right angles to the direction being considered

e.g.,

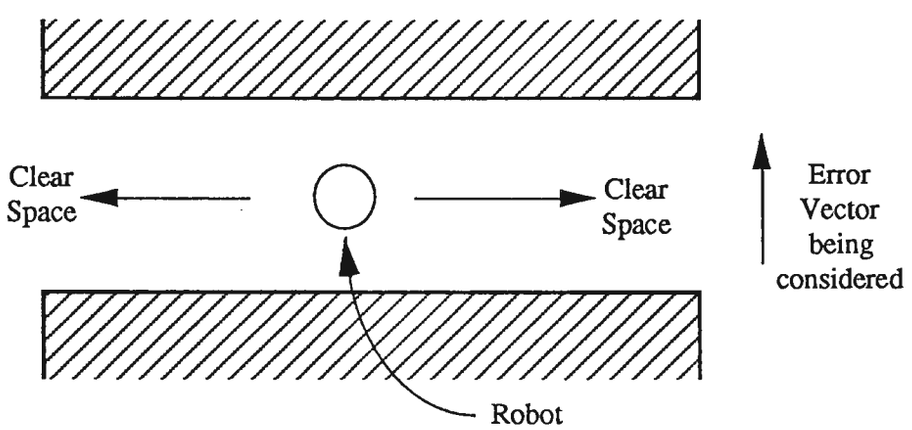


Fig 5.2 Test case to consider error in one direction only

This test used the map created in the first test with a 10 degree error introduced into the path vector (Fig 5.3). This should give an error vector after the path is traversed of 600 sin(10) or 104mm from point 1B.

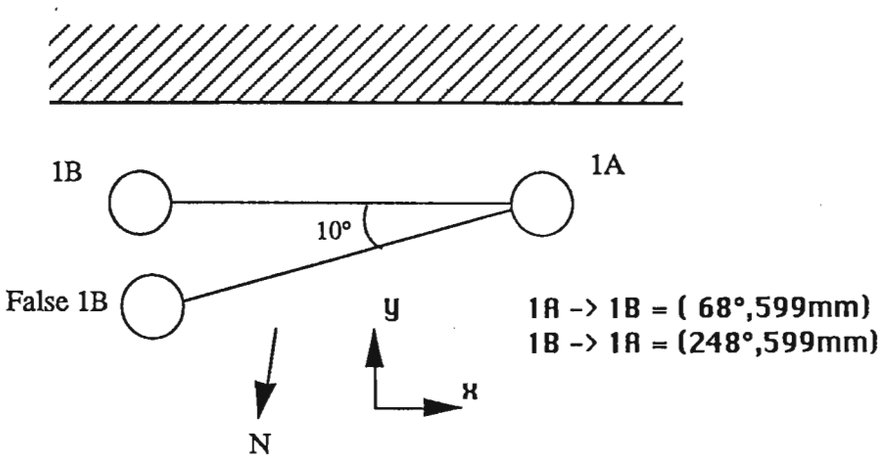


Fig 5.3 Introduced error in test map

This time the map was read into the navigation program. This program uses the same robot move commands as the Map Editor to move the robot from point to point. At each point it executes the full correction algorithm to re-position the robot on the target location.

The test was conducted by placing the robot at point 1A. The program was then instructed to move the robot from point 1A to 1B. This procedure was repeated three times and the results are listed in table 5.3.

Test Run	Point	Initial error to wall Y	Correction Angle	Distance	Final Error To wall Y X	Length
1	1B	-85mm	184°	95mm	-0mm	12mm
2	1B	-86mm	184°	94mm	-6mm	15mm
3	1B	-50mm	187°	93mm	-40mm	6mm

Table 5.3 Error Corrections in on Direction

On each test run the error result was less than expected, i.e., 85mm, 86mm and 50mm versus 104mm. No explanation for this could be determined. The angle of correction, 185° was expected as the path 1A -> 1B (68°,599) was not exactly parallel to the wall used. The wall

was at an angle of 88° giving a perpendicular error vector of 178° almost equal to the angle of correction.

5.1.3 Correction of accumulated error.

The third test was designed to show the the correction algorithm's ability to reduce accumulated error in one direction. The map used in this test had two path vectors at near to right angles connecting three points.

Point 1A was designated the start point and 1C as the target. The accumulated error of interest in this test is the error at 1C due to length error in path 1A -> 1B and the angle error between vectors 1A->1B and 1B->1C.

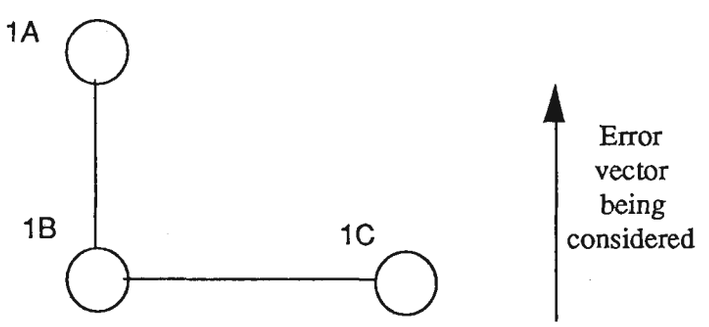


Fig 5.4 Direction of error being considered

The aim of the robot's software is to correct the length error in 1A -> 1B at 1B and then correct for the angle error at 1C. With no intermediate corrections the worst case error in this configuration is a positive error at 1B and a positive error at 1C accumulating to result in an error at 1C so large that it cannot be corrected. By controlling error at 1B the accumulated error may be manageable at 1C.

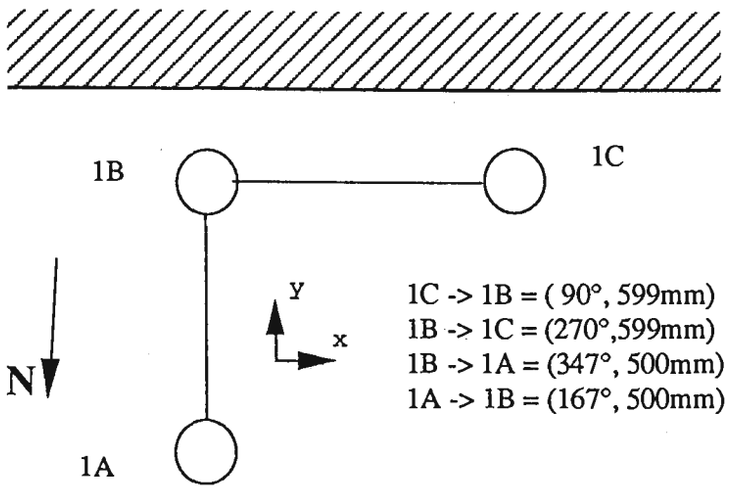


Fig 5.5 Second test map layout

The test map used in this test is shown in (Fig 5.5). Three runs of this test map were executed and the results are listed in table 5.4.

Test Run	Point	Initial error to wall Y	Correction Angle	Correction Distance	Final Error To wall Y	Final Error Length X
1	1B	0mm	185°	0mm	0mm	30mm
	1A	25mm	187°	29mm	4mm	40mm
2	1B	28mm	182°	3mm	-24mm	40mm
	1A	82mm	187°	123mm	41mm	44mm
3	1B	12mm	189°	0mm	-14mm	-20mm
	1A	32mm	184°	29mm	2mm	-12mm

Table 5.4 Accumulated Error Correction

As no error was introduced at any point the initial errors in this test were all small. Despite this, the correction process was able to improve the placement of the robot, closer to the synchronization point. In the third test run at point 1B the increase in error was unexpected. The increase was due to the robot spiralling off the point during the "scan" phase of the correction process.

The next test was included to demonstrate the software's resistance to error. This was done by using the same map as in the third test with angle error added at 1B.

The modified test map used is listed below

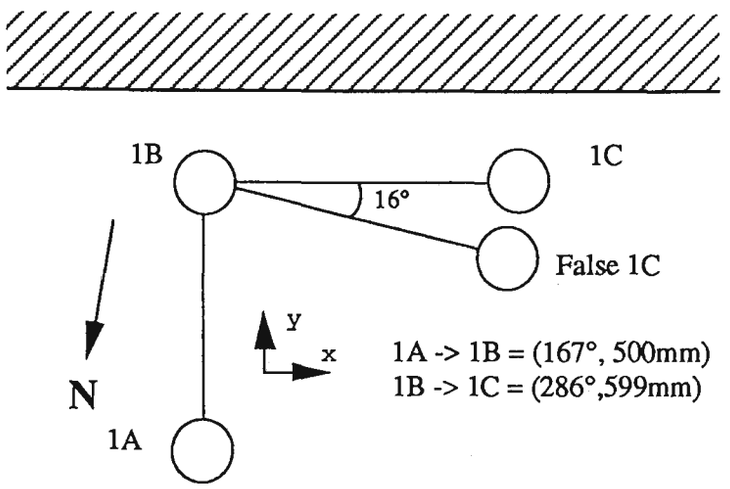


Fig 5.6 Error introduced in second test map

Three test runs of this map were executed and the results are listed below (Table 5.5).

Test Run	Point	Initial error to wall Y	Correction Angle	Distance	Final Error To wall Y	Length X
1	1B	0mm	185°	0mm	0mm	-21mm
	1C	-244mm	184°	233mm	-20mm	-4mm
2	1B	-17mm	184°	0mm	-17mm	-19mm
	1C	-306mm	189°	223mm	-82mm	25mm
3	1B	-12mm	189°	1mm	-10mm	10mm
	1C	-283mm	185°	223mm	-60mm	40mm

Table 5.5 Accumulated Induced Error Correction

An error to the wall of 165mm was expected. Each of the initial errors at 1C was greater than this. This contrasts with the results in Table 5.3 and may indicate some problem with the direction control sub-system. Once again the error correction algorithm was able to improve the position of the robot, moving it closer to the synchronization point.

5.1.4 Error Correction in Two Dimensions.

The final 3 tests were designed to demonstrate the correction algorithms ability to correct accumulated error in two directions. The map used in this test consisted of three paths connecting four points.

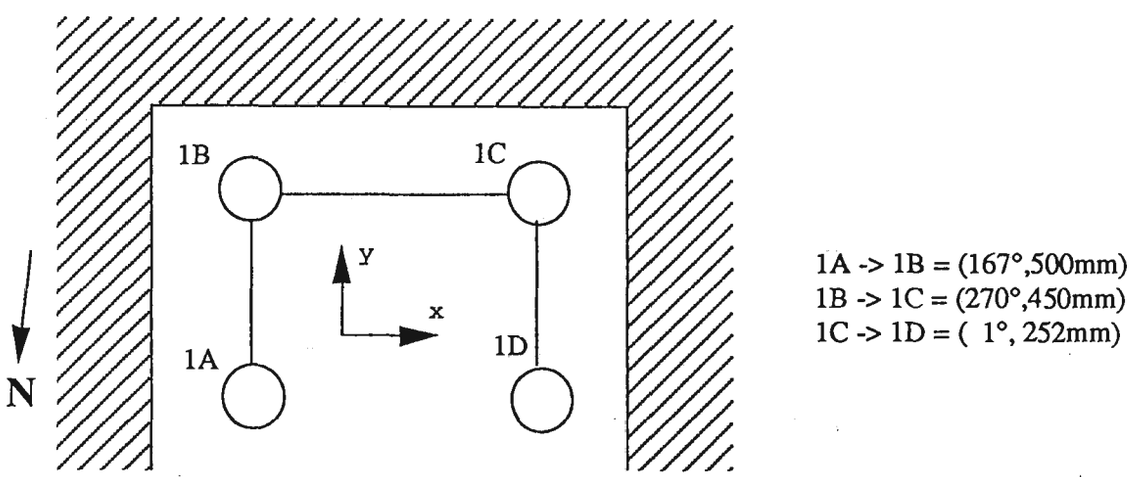


Fig 5.7 Final test map layout

Positional error was introduced at each point (see Fig 5.8) and the resultant expected error at 1D was 69mm in the Y direction and 152mm in the X direction. The error at 1D can be calculated as follows. Error in the Y direction is error in the length(1A -> 1B) plus, Length(1B -> 1C) * SIN(error in angle at 1B) plus error in length(1C -> 1D). In the X direction the error at 1D is error due to length(1A -> 1B) * SIN(error in angle at 1A) plus error in length(1B -> 1C) plus length(1C -> 1D) * SIN(error in angle at 1C).

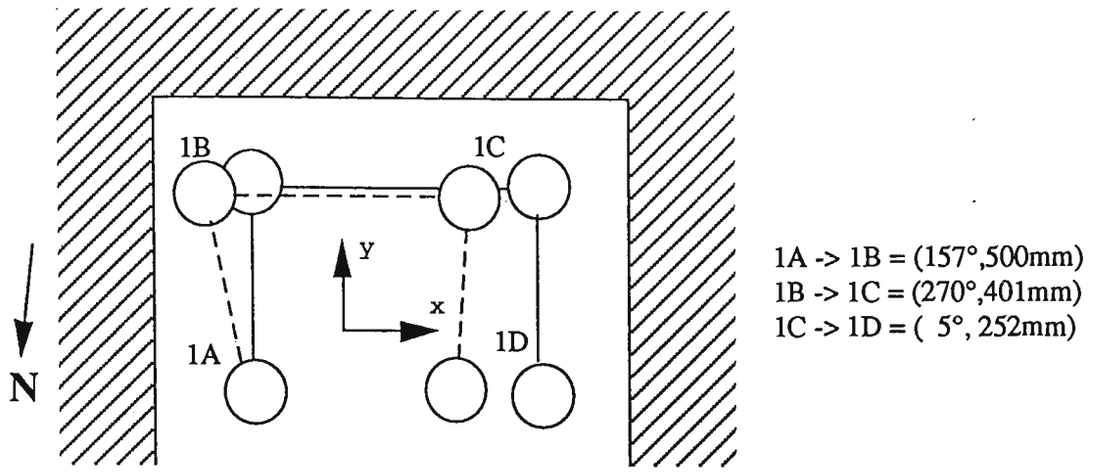


Fig 5.8 Introduced error in final test map layout

		No Error No Correction		Error No Correction		Error & Correction					
Test Run	Point	X	Y	X	Y	Initial Error X	Initial Error Y	Correction Angle	Correction Distance	Final Error X	Final Error Y
1	1B	3	-7	12	-97	-55	26	181°	24	-50	-2
	1A	8	0	-20	-152	-110	5	278°	86	-10	2
	1D	20	0	-20	-203	-37	10	273°	18	1	16
2	1B	0	-20	7	-45	-104	18	182°	22	-105	6
	1A	0	-15	3	-96	-155	-25	275°	142	11	-40
	1D	-3	-23	-8	-135	-30	-35	14	19	-6	-33
3	1B	-3	-13	4	-79	-75	30	185°	14	-72	3
	1A	-20	-17	-9	-126	-120	-10	278°	94	-14	-17
	1D	-21	-26	-9	-176	-65	-20	278°	35	-8	-12

Table 5.6 Accumulated Error Correction in Two Dimensions

Once again 3 test runs were performed and the results are listed in table 5.6. This final set of test results demonstrates the success of the platform in navigating from point to point within a controlled environment.

6. Conclusions

6.1 Aims Analysis

The test vehicle was constructed from inexpensive components. It is able to navigate in a known environment with certain constraints. A discussion of the degree to which each design aim was achieved follows.

6.1.1 Sensors

All sensors used were of simple design and cost effective. In the case of the infra-red sensors, several simple sensors (wheel movement, compass and infra-red intensity conversion) were combined to form a more complex unit under the control of the robots micro-controller.

6.1.2 View

The scan sensor was achieved using infra-red reflections. The signal was modulated and filtered at a frequency which avoided interference from most common infra-red sources. The received signal was converted to a digital value and through a conversion table the distance in mm was obtained. To perform a scan several of the robot's other sensor's results were combined to produce the scan results. Although this combination worked the use of an infra-red sensor that can independently perform a scan is strongly recommended. The performance for the cost was nonetheless very encouraging.

6.1.3 Distance

The distance travelled is measured using reflective strips on the wheels. A conversion factor was worked out so the operator can specify mm and the value converted to a number of strips to control the

robot's forward and circular movements. One strip equates to 4.5mm forward travel or 1.6 degrees rotation.

6.1.4 Direction

A simple compass was implemented. It is only able to sense north but all other directions can be deduced by controlling the rotation of the robot. The accuracy of deduced angles is limited to every 1.6 degrees by the wheel sensors.

6.1.5 Map Storage

As per requirements the robot did not explore its environment but instead the operator enters the details of the environment. Facilities are provided to store the map information to disk for further use. Once created, the map may also be modified and stored for re-use.

Whilst in use the map is stored in memory in three simple structures. The structures represent rooms, locations within rooms and paths between points. Maintaining the links between structures required some extra programming effort but the performance (in terms of speed of program execution) of the memory storage was extremely good.

6.1.6 Wall Detection

Walls were considered to be any straight lines identified within the local environment. A method of identifying walls was developed. The algorithm requires that the walls start and finish outside the local view. As walls that end within view are ignored, difficulties arise at outside corners as valuable information is being over-looked.

6.1.7 Position correction

The robot, under several test cases, demonstrated an ability to correct its position. It was only able to do this within an error size range of +/- 980mm. This constraint is due to the accuracy of the infra-red sensor at its limits.

Once the error was determined it was applied to the robot's position immediately. Once the robot is re-positioned (to the corrected location) the next step in the chosen route can be taken.

6.2 Uses in the real world

6.2.1 Unrealistic limitations

The usefulness of the test vehicle in the 'real' world is very limited. The test vehicle has no adaptors that would enable it to perform useful tasks, not even simple transport tasks. A redesign of the drive mechanism is required before the navigation algorithms could be used. The design assumptions (smooth floors, a known environment and the controller software specifically for the test vehicle) were made to simplify the construction of the vehicle and lead to the above problems.

The limitations that are more difficult to overcome in a production vehicle are those that specify environment factors. Those are the need for flat walls, no magnetic fields and a non-dynamic environment. In an industrial environment buildings tend to be large and are 'cluttered' with machinery that does not present a straight line reflection necessary for wall detection. Industrial environments are also riddled with electric motors and extensive power cabling producing many sources of magnetic interference which would adversely effect the compass readings.

The design's inability to handle dynamic environments renders it useless. This is best illustrated by the proposed use as a transport. The very payload (unless very small) once moved by the robot would

constitute a dynamic environment. It may be useful in a warehouse environment if it is possible to keep the payloads (both pick-up and placement) locations above the infra-red sensor levels. The robot could then navigate without the removal and placement of the payload changing the local environment.

6.2.2 Development of the Infra-red Sensor

Although time was spent on developing the infra-red sensor further improvements could be made. The areas that could be improved include the distance range, the mechanism of the scan and the use of the scan information.

6.2.2.1 Distance range

The configuration chosen aimed to minimise the effects of the relationship between light intensity and distance. A simpler sensor design with more emphasis on pre-processing the signal before conversion to digital may improve results. The use of optics to focus the transmitted infra-red beam and collect the returned reflection is a possibility that should be considered. Increasing the output of the infra-red beam would also improve the range. These possibilities and others should be considered to improve the sensor and the accuracy of the navigation process.

6.2.2.2 Robot stationary during scan

The saving in mechanism made by mounting the sensor on the robots frame and rotating the robot to achieve the scan effect caused more problems than the savings were worth. The movement of the robot caused errors to both the compass readings and the position. Once the compass is read the accuracy can only be guaranteed for one move or turn. The scan requires two more (to stabilise the power supply and to achieve full speed before recording infra-red sensor readings). The rotation on the spot using two separately driven wheels results in the platform crab walking, creating a positioning error during the correction process.

A mechanism that allows the robot to remain stationary and moves the infra-red sensor through 360 degrees would over-come both problems. The time saving would also be significant. It would avoid one of the find north commands issued by the scan control program.

6.2.2.3 Use of scan information

The use of all the scan information will improve the robot's ability to correct its position. As identified in the wall detection section, an increase in processing power would be required to conduct a full feature analysis of the scan results. Likewise, to use all the available walls to construct the error vector would require extra processing power.

In the calculation of the error vector a compromise that would significantly improve the accuracy of the error vector would be to consider one extra wall. If a second wall were chosen that was as close as possible to right angles to the first wall it would include the error component that is currently missing.

6.2.2.4 Half walls

No considerations were given to the possibility of walls ending within the view of the robot.

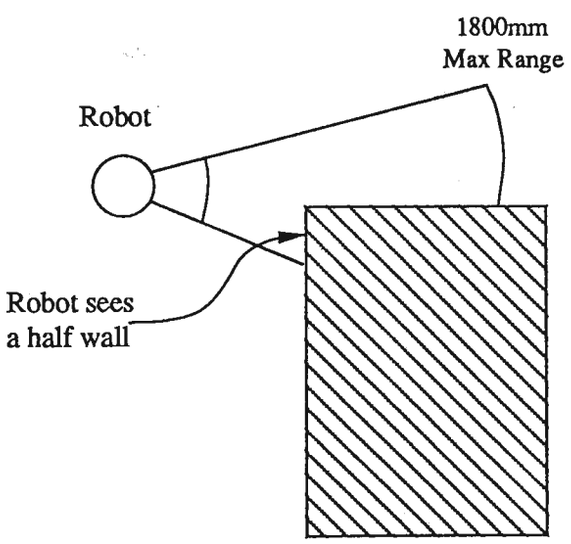


Fig 6.1 Detecting half walls

The inclusions of this information in the point comparison would further increase accuracy. The pattern for a half wall is the same as a full wall, but no symmetry is observed. This feature analysis would be simple to include.

6.2.3 Ability to correct

The test vehicle has a infra-red sensor that has a range of 100mm to 1800mm. At first this appears small but most rooms are smaller than 3.6m across. Also open spaces are uncommon in rooms, most are broken up by objects that can be just as useful to navigate by. Large spaces can be planned for by selecting synchronization points close to the perimeter. The journey around the edge of a large room may be twice as long but the positioning of the robot at the end will have a smaller error content compared to the position after an expedition across the centre.

The appearance of objects that are not flat walls will not greatly impact the accuracy. If the object was not previously there it will be ignored as it will not be matched in the old scan or the wall detection algorithm will reject it as not a wall (shape permitting). The most significant impact is from foreign objects placed between the robot and a close known wall. In this case the wall detection algorithm will reject that wall and correction will fail to occur.

The accuracy of the test vehicle is very dependant on the dynamics of the environment and the design and placement of synchronization points. In practice the careful placement of synchronization points is not an unreal restriction. If care in synchronization point placement is required to produce accurate results then that is a requirement for the operator.

6.2.3.1 Angle to Wall Tolerance

The point comparison algorithm detects walls by searching for the straight line pattern. Once the scans have been searched for line

patterns the two lists of walls (lines) are compared. The comparison matches walls by the angle to the wall. To provide some tolerance to differences in the north reference point between the current and the stored scans an allowance of +/- 10 degrees has been made. The angle range used to conform a wall is +/- 32 degrees so despite a 10 degree error the wall in question must be the same wall, as the wall conform range overlaps in the two scans.

The allowance for rotation should also be used as a confidence factor when selecting walls for use in calculating the position offset error vector. If the error is truly due to rotation of the robot when recording the scans all walls detected will have the same rotation confidence factor and they will all cancel each other out.

6.2.3.2 Not All Walls Compared.

The wall detection algorithm returns a list of all the walls within a scan. The point compare algorithm searches this list in a pre-defined sequence. It uses the first wall it finds that meets the selection criteria to calculate the error vector. As defined in the point compare section this scheme ignores any component of the error that is parallel to the detected wall.

Significant improvements are possible to the point compare algorithm by using all the walls detected in the calculation of the error vector.

6.2.3.3 Foreign Objects

The available information about foreign objects is only useful if the information can be extracted from the scan. To do this would require pattern analysis of the scan to give details of the shape and distance to the object. The complexity of classifying previously unseen object scans appeared to require more processing capacity than available.

An approach that was considered was to work out the displacement of the robot in terms of flat walls. Using the displacement work out what the scan should have looked like and compare that with what was recorded. Any differences should be due to foreign objects. As the calculation of the exact displacement was not done this method was not tried and the extra available information was wasted.

6.2.3.4 Error Vector Shortcomings

The algorithm for calculating the error vector only uses the largest distance error to a wall. This approach is fine if the error is in that direction or no other walls can be seen. Where more than one wall can be seen the error vectors calculated from the displacement to all walls should be considered.

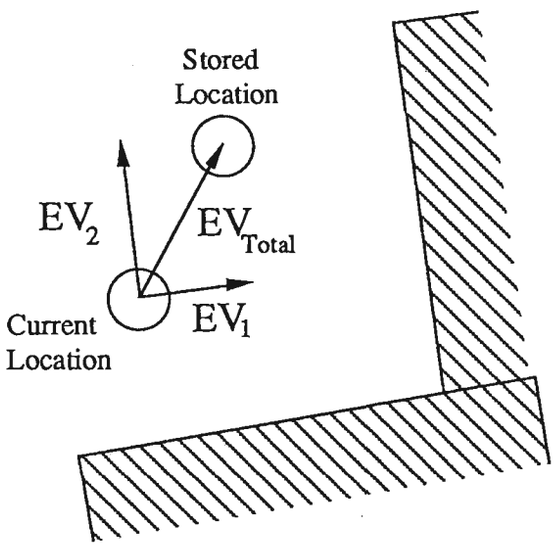


Fig 6.2 Calculation of error vector

To calculate the total error vector the largest error vector should be selected. All the other error vectors should be normalised to the first and then added to it to give a total error.

6.2.3.5 Error Vector Error Evaluation.

An evaluation of error inherent in the error vector's calculation is as follows.

Let Pa = Point Actual (stored scan)
Pb = Point Current (current scan)
Ea = Actual Error

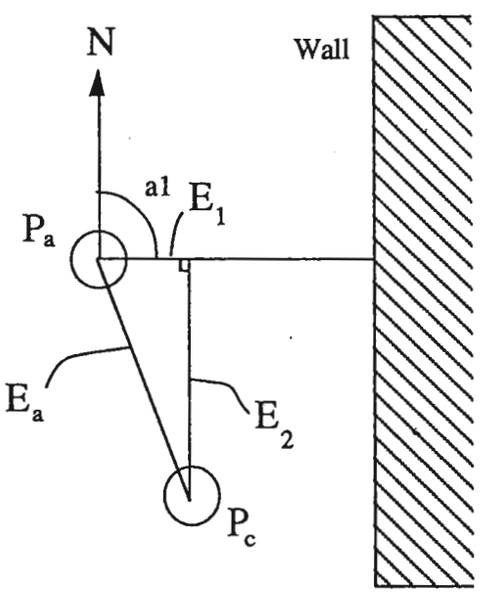


Fig 6.3 Calculation of error vector

Ea = Pa - Pc

From the algorithm used to calculate offset error Ea can be redefined as

Let Ea = E1 + E2

Where

E1 = {l1,a1} Error vector calculated
E2 = {l2,a1+/- 90} Error not included by algorithm
l is the error vector's length.
a is the angle of the error vector.

The error vector algorithm selects the largest error to a wall:

if there is a wall between a1+90 and a1-90 degrees the size of the error is:

$$| l2 | < | l1 |$$

if there is no wall between a1+90 and a1-90 degrees or the wall at that angle is out of range then the size of the error is:

$$0 <= | l2 | < | \text{largest room size} |$$

Where no wall occurs at a1 +/- 90 degrees the size of the error component at 90 degrees to the detected error is indeterminate. This is the case when the robot is placed in a hall, two sides close and two distant. An error vector can be calculated from the two close walls. As the two distant wall are out of range no comparison between expected position and recorded position is possible. As no comparison is possible a large error may exist but will remain undetected.

6.2.3.6 Fixed Scan Slice for Wall Detection.

Walls are detected by searching the scan results for the wall pattern. Using the current algorithm the longest detectable wall is 235mm (from 32 degrees either side of the robot) when the robot is 100mm from the wall. At the other limit (1000mm) the maximum length becomes 2358mm. As the robot can distinguish distances between 100mm to 1000mm the view of objects is between 235mm and 2358mm wide. This shows that the closer the wall the lower the confidence level is that the wall is flat. The wall detect algorithm needs to be able to adjust to the distance to the wall and not just use a blanket 64 degrees wide slice of the scan to prove a wall in.

6.2.3.7 Walls Must Be Close

The maximum range of the infra-red sensor on the test vehicle is 1800mm. The sensor works well between 100mm and 500mm. At 500mm an accuracy of +/- 25 mm is recorded where as the accuracy drops to +/- 50 mm at 1000mm.

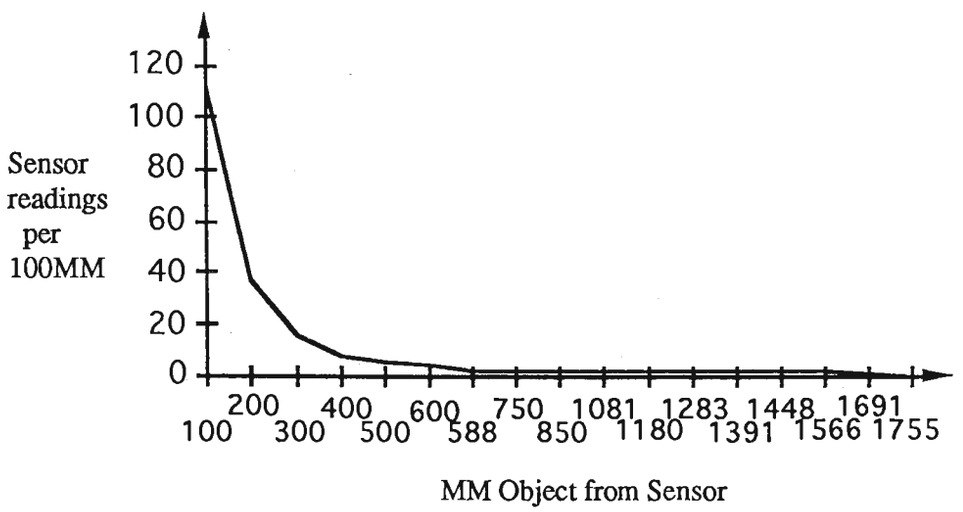


Fig 6.4 Sensor readings Vs distance

The requirement of being between 100mm and 1000mm from a wall in order to be able to correct is very restrictive. In the situation where a target location is in the centre of a room (more than 1000mm from a wall) no accuracy of placement can be guaranteed.

6.2.4 Method of Correction

Once the error vector is calculated the system goes about correcting the position of the robot. It does this by generating and issuing the commands necessary to instruct the robot to move to the corrected point location.

This situation can be improved by repeatedly performing the correction process. This would involve scanning again once the corrected location was reached. This scan would then be compared with the stored expected scan. This comparison would produce an error vector that could be applied by moving the robot according to the second error vector. This process could be repeated until the resultant error vector dropped to a zero offset.

The improvements achieved by repeating the correction process would be effected by two factors. The first is due to new errors induced by the moves necessary to apply the error vectors calculated by each attempt. The second factor would be the effect of dynamic

environments. One of the strengths of making one correction based on the largest error detected is that if the change in the wall position is due to foreign objects it will be corrected at the next point. This is because the object is unlikely to be large enough to impact the scan at the next point.

6.2.4.1 Correction on the fly

An alternative to applying the error vector as soon as it is calculated is to add it to the next path's vector. This would avoid the error induced by moving the robot to correct the initial error. This approach was not tried due to two possible problems. The first is that to add the error vector to the next path would require an error vector containing components from more than one wall offset. If only one was considered then that error part of the error will still be evident in at the next point as well as containing new errors introduced due to the move.

The second problem would be the possibility that the resultant path would lead the robot through a wall or a corner as demonstrated in the example below.

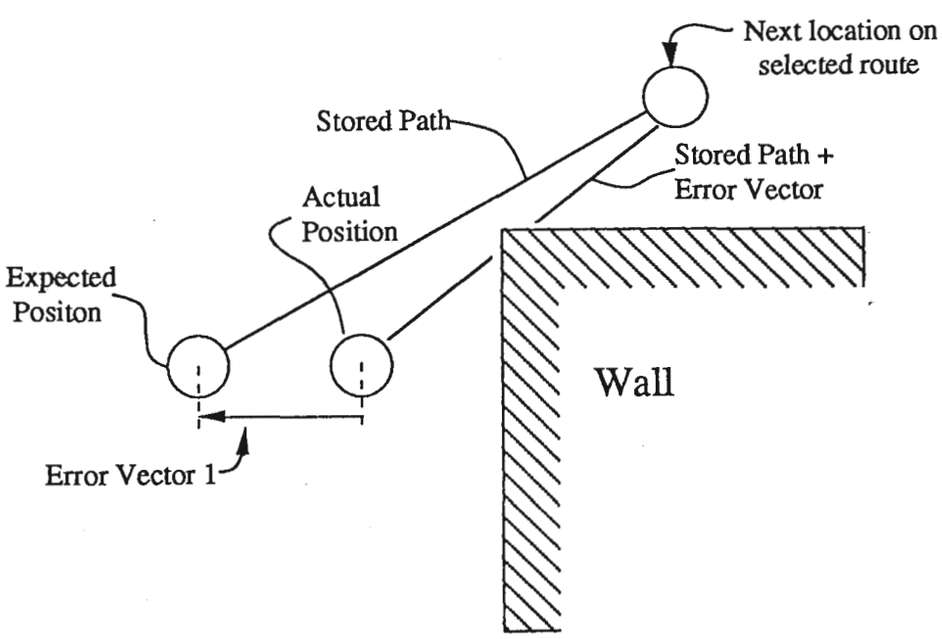


Fig 6.5 Blocked path due to positional error

7. References

[1] Bentham J. and Bentham F., "Ultrasonic Vision System", Practical Electronics, pp24-28, pp 66-67, April / May 1982

[2] Brooks, R. A., "Solving the Find Path Problem by a Good Representation of Free Space", IEEE Transactions on Systems, Man and Cybernetics, SMC-13 No.3, March 1983

[3] Chatita R. and Laumond J., "Position Referencing and Consistent World Modelling for Mobile Robots", IEEE 1985 International Conference on Robotics and Automation, March 1985

[4] Clayton G.B., "Operational Amplifiers", Newnes - Butterworth's Cox and Wyman Ltd, Section Edition,pp321-322

[5] Da Casta F., "How to Build Your Own Working Robot Pet"
TAB. 1979

[6] Everett H. R., "Survey of Collision Avoidance and Ranging Sensors for Mobil Robots", Robotics and Autonomous Systems, Vol. 5. 1989

[7] Everett H. R. and Flynn A. M., "A Programmable Near-Infrared Proximity Detector for Mobile Robot Navigation", Proceedings of the SPIE Mobile Robot Workshop, Cambridge, Massachusetts. October 1986

[8] Frywell R. C., "Navigation Planning Using Quadtrees: Proceedings of SPIE, The International Society for Optical Engineering", Mobile Robots II, Vol 852, Cambridge, Massachusetts, 1987

[9] Harmon S. Y., "The ground Surveillance Robot (GSR): An Autonomous Vehicle Designed to Transit Unknown Terrain", IEEE Journal of Robotics and Automation, Vol. RA-3 No. 6, December 1987

[10] Hille R.F., "Data Abstraction and Program Development using Pascal", Prentice Hall, 1988

[11] Intel Corporation,"MCS-51 Macro Assembler User's Guide",
Intel Corporation Literature Department, 1981

[12] Kabuka M.R. and Arenas A. E., "Position Verification of a Mobile Robot Using Standard Pattern", IEEE Journal of Robotics and Automation, Vol RA-3 No.6, December 1987

[13] Karayama Y. and Yata S., "Vehicle Path Specification by a Sequence of straight Lines", IEEE Journal of Robotics and Automation, Vol 4 No. 3, June 1988

[14] Kerigman D. J., Triendl E. and Binford T. O., "A Mobil Robot: Sensing, Planning and Locomotion", IEEE 1987 International Conference on Robotics and Automation Standford University, California, 1987

[15] Kernighan B.W. and Ritchie D.M., "The C Programming Language", Prentice Hall, 1982

[16] Koch E., Yeh C., Hillel G., Meystel A. and Isik C. "Simulation of Path Planning for a System with Vision and Map Updating", IEEE 1985 International Conference on Robotics and Automation, March 1985

[17] Lof G.W. and Chen P.I., "Design of an Automatic Guided Vehicle, 17th International Symposium on Industrial Robots", Chicago Illinois, April 1987

[18] MicroSoft Corporation, "PC 10/20/40 a2000 MS-DOS 3.2", pp 257-262, pp26-116, April 1987

[19] Russell R.A., "Design of a Micro Computer-Based LOGO Turtle Interface", Microprocessors and Microsystems, Butterworth & Co Ltd, Vol 7 No 2 March 1983

[20] Russell R.A. and Wyse D., "D.A.I. Technical Memo 16", University of Endinburgh, pp4 & 7, May 1981

[21] Schwartz J. T. and Sharir M. "On the Piano Movers Problem, the case of a Two Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers", Communications on pure and Applied Mathematics, Vol. 36, No 5 May 1986

[22] Singh, J.S. and Waugh, M.D. "Robot Path Planning Using Intersecting Convex Shapes Analysis and Simulation", IEEE Journal of Robotics and Automation,Vol RA-3 No. 2 April 1987

[23] Siy P., "Road Map Production System for Intelligent Mobile Robot",
PhD dissertation, Department of Electrical and Computer Engineering, Wayne State University, Detroit, Michigan

[24] Steer B. "Trajectory Planning for Mobil Robots", The International Journal of Robotic Research, Vol 8 No. 5, Oct 1989

[25] Zelinsky A. "Robot Navigation with Learning", The Australian Computer Journal, Vol. 20 No. 2, May 1988

Appendix

A.1 Map Storage Records

```
/* links between points memory storage record */

Struct routes   {           int         dist ;
                int         angle ;
                struct location  *nxt_loc ;
                struct routes    *others ;
              } ;


/* known points memory storage record */

struct location { char      name[3] ;
                int         rm ;
                struct location  *nxt_loc ;
                int         on_path ;
                int         scan[218] ;
                struct routes    *path ;
              } ;


/* room list memory storage record */

struct rooms    {           int         rm ;
                struct rooms     *nxt_rm ;
                struct location  *loc ;
              } ;

/* location file storage record */

struct rec_a  { char        type ;
                char        name[3] ;
                int         rm ;
                int         scan[218] ;
              } ;

/* route file storage record */

struct rec_b  { char        type ;
                int         from_rm ;
                char        from_name[3] ;
                int         to_rm ;
                char        to_name[3] ;
                int         dist ;
                int         angle ;
              } ;
```

A.2 Source Code

A.2.1 Wall Detection

```
int line_find( struct location *point, struct lines *lines )

{  int i, p, count, status ;

   d_scan[ 0 ] = Point -> scan[ 0 ] - Point -> scan[217 ] ;
   for ( i = 1 ; i < 218 ; i++ )
     {  d_scan[ i ] = Point -> scan[ i ] - Point -> scan[ 217 ] ;
        dd_scan[ i ] = d_scan[ i ] - d_scan[ i - 1 ] ;
     }
   dd_scan[ i ] = d_scan[ i ] - d_scan[ i - 1 ] ;

   wall_no 0 ;
   for( p = 0 ; p < 218 ; p++ )
     if ( Point -> scan[p] < open_space && d_scan[ i - 1 ] == 0 )
       { i = top_of_peak( p, Point ) ;
         store_peak( i, Point, Lines ) ;
         if ( wall_no >= 20 ) break ;
         p = end_of_peak( p, Point ) ;
       }

   Lines -> number = wall_no ;
}

static int top_of_peak( int p, struct location *Point )
{  int i, j ;
   for( i = 0 ; i < 40 ; i++ )
     {  j = p + i ;
        if ( j >= 218 ) j = j - 218 ;
        if ( Point -> scan[p] != Point -> scan[ j ] ) break ;
     }

   j = p + i /2 ;
   if ( j >= 218 ) j = j - 218 ;
   return( j) ;
}
```

```
static int store_peak( int i, struct location *Point, struct lines *lines
{  int j, k, l, m, n ;
   for( j = 1 ; j < 20 ; j++ )
      {  k = i + j ;
         if ( k > 218 ) k = k - 218 ;
         l = i - j ;
         if ( l < 0 ) l = l + 218 ;
         if ( dd_scan[ k ] > 0 && dd_scan[ l ] < 0 ) break ;
         if ( dd_scan[ k ] < 0 && dd_scan[ l ] > 0 ) break ;
      }

   if ( j > 10 )
      {  Lines -> walls[ wall_no ].ang_wall = i ;
         Lines -> walls[ wall_no].wsize = j ;
         Lines -> walls[ wall_no++].wall90 = Point -> scan[ i ] ;
      }
}

static int end_of_peak( int p, struct location *Point )
{  init i, j ;

   for( i = 0 ; i < 40 ; i++ )
      {  j = p + i ;
         if ( j >= 218 ) break ;
         if ( Point -> scan[ p ] != Point -> scan[ j ] ) break ;
      }
}
```

A.2.2 Error Vector Calculation

```
int  delta_loc( error, now, map )
struct error_vec      *error ;
struct location       *now ;
struct location       *map ;

{  int i, j, ;
   int max_ang, min_ang ;
   int  max_dist, min_dist ;
   int error_angle ;
   int max_lines, ang_used ;
   double ang ;

   error -> dist = 0 ;
   error_angle = 0 ;

 /* find the walls in the current and expected scans */

   line_find( now, &lines_now ) ;
   line_find( map, &lines_map ) ;

   for( i= 0 ; i < lines_map.number ; i++ )
     {  /* calcualte an error range to match lines with */

        max_ang = lines_map.walls[ i ]. ang_wall + 6 ;
        if ( max_ang >= 218 ) max_ang = max_ang - 218 ;
        min_ang = lines_map.walls[ i ].ang_wall - 6 ;
        if ( min_ang < 0 ) min_ang = min_ang + 218 ;

        /* for every line found in the current scan try and match it */
        /* to the line found in the stored/expected scan */
        for( j = 0 ; j < lines_now.number ; j++ )
          if ( lines_now.walls[ j ].ang_wall > min_ang &&
               lines_now.walls[ j ].ang_wall < max_ang ) break ;

        if ( lines_now.walls[ j ].ang_wall > min_ang &&
             lines_now.walls[ j ].ang_wall < max_ang )
          { /* if a line is found in both calc an error vector for that lin
            error -> dist = lines_now.walls[ j ].wall90 -
                            lines_now.walls[ i ].wall90 ;
            error -> angle = lines_now.walls[ j ].walls[ j ].ang_wall ;
```

```
            max_dist = lines_map.walls[ i ].wall90 + 10 ;
            min_dist = lines_map.walls[ i ].wall90 - 10 ;
            if ( min_dist < 0 ) min_dist = 0 ;
            /* if the matched line (expected and current) have an erro
            /* larger than the
            if ( lines_now.walls[ j ].wall90 > max_dist ||
                 lines_now.walls[ j ].wall90 < min_dist )
              break ;
          }
      }
    if ( error -> dist < 0 )
      {  error -> dist = - error -> dist ;
         error_angle = error_angle + 109 ;
         if ( error_angle >= 218 ) error_angle = error_angle - 218 ;
      }

    error_angle = tics_to_degs( error_angle ) ;
}
```

A.2.3 Shortest Path Search

```
shortest()

{  int cost, deep, dist, angle, i ;
   cost = 0 ;
   deep = 0 ;
   dist = 0 ;
   angle = 0 ;

   bestcost = 30000 ;
   bestdeep = 0 ;

   walk( snode, cost, dist, angle )

   if ( bestcost == 30000 )
      error = "No path found from start to finish" ;
}

walk( node, cost )

struct location *node ;
int cost ;
{  struct routes *nxt ;
   int i ;

   list.point[ deep ] = node ;

   if ( node != fnode )
      {  /* if not at the finish point search all this points paths */
         node -> on_path = TRUE ;
         nxt = node -> path ;
         while( nxt != NULL )
           {  if ( ( nxt -> loc ) -> on_path == FALSE )
                 if ( cost + nxt -> dist < bestcost )
                   { list.angle[ dep ] = nxt -> angle ;
                     list.dist[ dep ] =nxt-> dist ;
                     deep++ ;
                     walk( nxt->loc, cost + nxt -> dist ) ;
                     deep-- ;
                   }
              nxt = nxt -> others ;
           }

         node -> on_path = FALSE ;
      }
```

```
  else
    {  /* this is the finish location, check if it is the best route */

       if ( cost < best_cost )
         {  /* save the best route for later reference */
           for ( i = 0 ; i <= deep ; i++ )
             {   best.point[ i ] = list.point[ i ];
                 best.point[ i ] = list.angle[ i ] ;
                 best.dist[ i ] = list.dist[ i ] ;
             }
           bestcost = cost ;
           bestdeep = dep ;
         }
    }
}
```

A.2.4 PC Smoothing Algorithm

```
int scan_results( int view[218 ] )
{  int i ;

   for( i = 0 ; i < 218 ; i++ )
      if ( view[ i ] < 256 && view[ i ] > 0 )
         view[ i ] = mv_mm[ view[ i ] ] ;

   for( i = 0 ; i < 16 ; i++ )
      {  smooth_pass_1( view ) ;
         smooth_pass_2( view ) ;
      }
}

smooth_pass_1( int view[218] )
{  int i, j, k ;
   i = 216 ;
   j = 217 ;
   for( k = 0 ; k <= 217 ; k++ )
      {  view[ j ] = ( view[i] + view[j] + view[j] + view[k] + 3) / 4 ;
         i = j ;
         j = k ;
      }
}
smooth_pass_2( int view[218] )
{  int i, j, k ;
   i = 1 ;
   j = 0 ;
   for( i = 217 ; i > 0 ; i-- )
      {  view[ j ] = ( view[i] + view[j] + view[j] + view[k] + 3) / 4 ;
         k = j ;
         j = i ;
      }
}
```

A.2.5 Micro-Controller Smoothing Algorithm


```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     580
                     581 ;-------------------------------------------------;
                     582 ;           SMOOTH THE RESULTS OF THE I/R SCAN.   ;
                     583 ;-------------------------------------------------;
                     584 ;
                     585             GLOBAL  S_SCAN
                     586
    0278 120000      587 S_SCAN:     CALL    GET_CHR
                     588
    027B 120282      589             CALL    SMOOTH
                     590
    027E 120000      591             CALL    STATUS
    0281 22          592             RET
                     593
                     594 ;-------------------------------------------------;
                     595 ;           SMOOTH 4 TIMES USING BOTH ALGOs.      ;
                     596 ;-------------------------------------------------;
                     597 ;
    0282 7C04        598 SMOOTH:     MOV     R4,#4
                     599
    0284 12028D      600 SMOOTH1:    CALL    PASS_1
                     601
    0287 1202F0      602             CALL    PASS_2
                     603
    028A DCF8        604             DJNZ    R4,SMOOTH1
                     605
    028C 22          606             RET
                     607
                     608 ;-------------------------------------------------;
                     609 ;           FIRST PASS SMOOTHING.                 ;
                     610 ;-------------------------------------------------;
                     611 ;
    028D             612 PASS_1:
                     613 ;           FIRST PASS  OF X(I) = (X(I-1) + 2*X(I) + X(I+1))/4
                     614
    028D 7D00        615 SS0:        MOV     R5,#0
    028F 7E01        616             MOV     R6,#1
    0291 7F02        617             MOV     R7,#2
                     618
    0293 8D00        619 SS1:        MOV     REG_0,R5
    0295 75A000      620             MOV     P2,#0           ; Store the I-1 element.
    0298 E2          621             MOVX    A,@R0
    0299 FA          622             MOV     R2,A
    029A 75A001      623             MOV     P2,#1
    029D E2          624             MOVX    A,@R0
    029E FB          625             MOV     R3,A
                     626
    029F 8E00        627             MOV     REG_0,R6
    02A1 75A000      628             MOV     P2,#0           ; Add in the Ith element.
    02A4 E2          629             MOVX    A,@R0
                     630
    02A5 2A          631             ADD     A,R2
    02A6 FA          632             MOV     R2,A
                     633
    02A7 75A001      634             MOV     P2,#1
    02AA E2          635             MOVX    A,@R0
    02AB 3B          636             ADDC    A,R3
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE
    02AC FB           637         MOV     R3,A
                      638
    02AD 8E00         639         MOV     REG_0,R6
    02AF 75A000       640         MOV     P2,#0       ; Add the Ith element twice.
    02B2 E2           641         MOVX    A,@R0
                      642
    02B3 2A           643         ADD     A,R2
    02B4 FA           644         MOV     R2,A
                      645
    02B5 75A001       646         MOV     P2,#1
    02B8 E2           647         MOVX    A,@R0
    02B9 3B           648         ADDC    A,R3
    02BA FB           649         MOV     R3,A
                      650
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     652
    02BB 8F00        653          MOV     REG_0,R7
    02BD 75A000      654          MOV     P2,#0          ; Add in the I + 1 Element.
    02C0 E2          655          MOVX    A,@R0
                     656
    02C1 2A          657          ADD     A,R2
    02C2 FA          658          MOV     R2,A
                     659
    02C3 75A001      660          MOV     P2,#1
    02C6 E2          661          MOVX    A,@R0
    02C7 3B          662          ADDC    A,R3
                     663
    02C8 C3          664          CLR     C              ; Divide the result by 2.
    02C9 13          665          RRC     A
    02CA FB          666          MOV     R3,A
                     667
    02CB EA          668          MOV     A,R2
    02CC 13          669          RRC     A
    02CD FA          670          MOV     R2,A
                     671
    02CE C3          672          CLR     C              ; Divide the result by 2 again.
    02CF EB          673          MOV     A,R3
    02D0 13          674          RRC     A
    02D1 FB          675          MOV     R3,A
                     676
    02D2 EA          677          MOV     A,R2
    02D3 13          678          RRC     A
                     679
    02D4 8E00        680          MOV     REG_0,R6       ; Store the result in the i th element.
    02D6 75A000      681          MOV     P2,#0
    02D9 F2          682          MOVX    @R0,A
                     683
    02DA 75A001      684          MOV     P2,#1
    02DD EB          685          MOV     A,R3
    02DE F2          686          MOVX    @R0,A
                     687
    02DF 0D          688          INC     R5             ; Increment the loop counters.
    02E0 0E          689          INC     R6
    02E1 0F          690          INC     R7
                     691
    02E2 BED A02     692          CJNE    R6,#CIRCLE,SS2
    02E5 7E00        693          MOV     R6,#0
                     694
    02E7 BFDA02      695 SS2:     CJNE    R7,#CIRCLE,SS3
    02EA 7F00        696          MOV     R7,#0
                     697
    02EC BDDAA4      698 SS3:     CJNE    R5,#CIRCLE,SS1
                     699
    02EF 22          709          RET
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     703 ;
                     704 ;-------------------------------------------------;
                     705 ;          SECOND PASS SMOOTHING.                 ;
                     706 ;-------------------------------------------------;
                     707 ;
     02F0            708 PASS_2:
                     709 ;          SECOND PASS THIS TIME OF X(I) = (X(I-2) + X(I) + X(I+2))/4
                     710 ;
     02F0 7D00       711            MOV     R5,#0
     02F2 7E02       712            MOV     R6,#2
     02F4 7F04       713            MOV     R7,#4
                     714
     02F6 8D00       715 SS4:       MOV     REG_0,R5        ; Store the I-2 th element.
     02F8 75A000     716            MOV     P2,#0
     02FB E2         717            MOVX    A,@R0
     02FC FA         718            MOV     R2,A
     02FD 75A001     719            MOV     P2,#1
     0300 E2         720            MOVX    A,@R0
     0301 FB         721            MOV     R3,A
                     722
     0302 8E00       723            MOV     REG_0,R6        ; Add in the I th element.
     0304 75A000     724            MOV     P2,#0
     0307 E2         725            MOVX    A,@R0
                     726
     0308 2A         727            ADD     A,R2
     0309 FA         728            MOV     R2,A
                     729
     030A 75A001     730            MOV     P2,#1
     030D E2         731            MOVX    A,@R0
     030E 3B         732            ADDC    A,R3
     030F FB         733            MOV     R3,A
                     734
     0310 8E00       735            MOV     REG_0,R6        ; Add in the I th element twice.
     0312 75A000     736            MOV     P2,#0
     0315 E2         737            MOVX    A,@R0
                     738
     0316 2A         739            ADD     A,R2
     0317 FA         740            MOV     R2,A
                     741
     0318 75A001     742            MOV     P2,#1
     031B E2         743            MOVX    A,@R0
     031C 3B         744            ADDC    A,R3
     031D FB         745            MOV     R3,A
                     746
     031E 8F00       747            MOV     REG_0,R7        ; Add in the I + 2 nd element.
     0320 75A000     748            MOV     P2,#0
     0323 E2         749            MOVX    A,@R0
                     750
     0324 2A         751            ADD     A,R2
     0325 FA         752            MOV     R2,A
```

```
LOCATION OBJECT CODE LINE      SOURCE LINE

                     759
     032B C3         760              CLR     C              ; Divide the result by 2.
     032C 13         761              RRC     A
     032D FB         762              MOV     R3,A
                     763
     032E EA         764              MOV     A,R2
     032F 13         765              RRC     A
     0330 FA         766              MOV     R2,A
                     767
     0331 C3         768              CLR     C
     0332 EB         769              MOV     A,R3
     0333 13         770              RRC     A
     0334 FB         771              MOV     R3,A
                     772
     0335 EA         773              MOV     A,R2
     0336 13         774              RRC     A
                     775
     0337 8E00       776              MOV     REG_0,R6
     0339 75A000     777              MOV     P2,#0
     033C F2         778              MOVX    @R0,A
                     779
     033D 75A001     780              MOV     P2,#1
     0340 EB         781              MOV     A,R3
     0341 F2         782              MOVX    @R0,A
                     783
     0342 0D         784              INC     R5
     0343 0E         785              INC     R6
     0344 0F         786              INC     R7
                     787
     0345 BEDA02     788              CJNE    R6,#CIRCLE,SS5
     0348 7E00       789              MOV     R6,#0
                     790
     034A BFDA02     791 SS5:         CJNE    R7,#CIRCLE,SS6
     034D 7F00       792              MOV     R7,#0
                     793
     034F BDDAA4     794 SS6:         CJNE    R5,#CIRCLE,SS4
                     795
     0352 22         796              RET
                     797
```

A.3 Robot Structure Charts

A.3.1 Command Handler

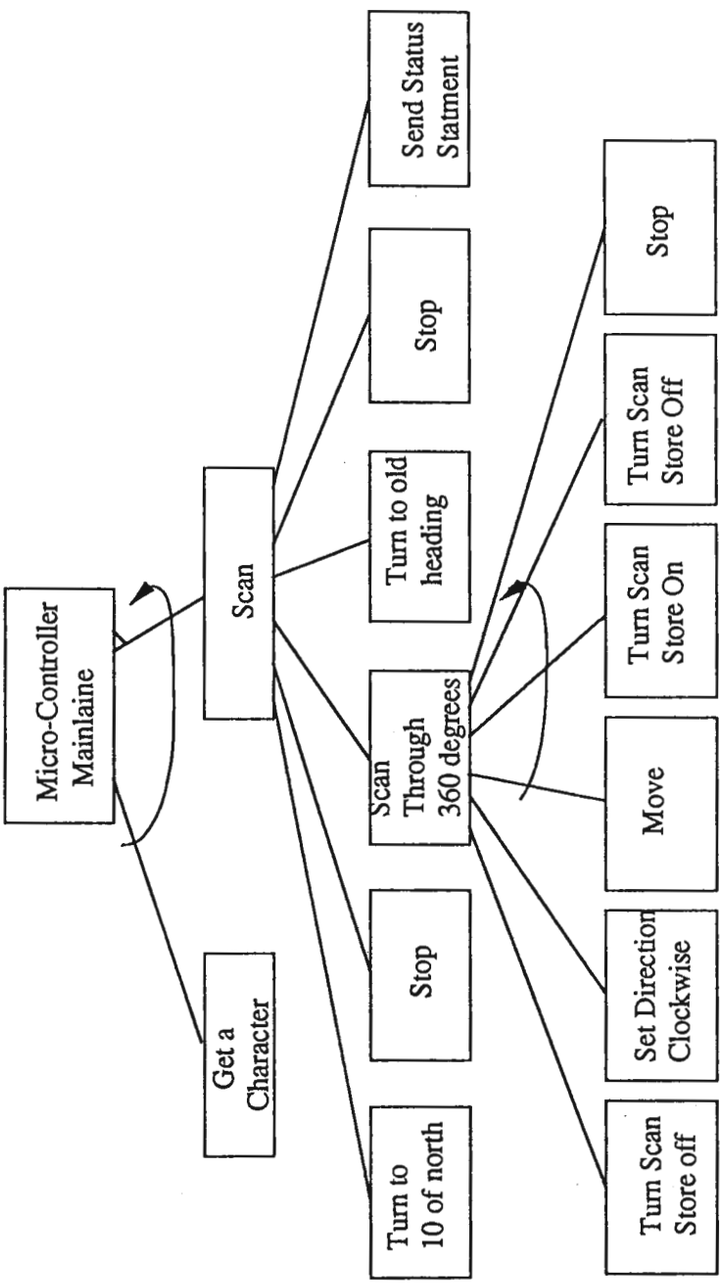


Fig A.1 Structure chart of scan command subroutine

A.3.2 Forward Command

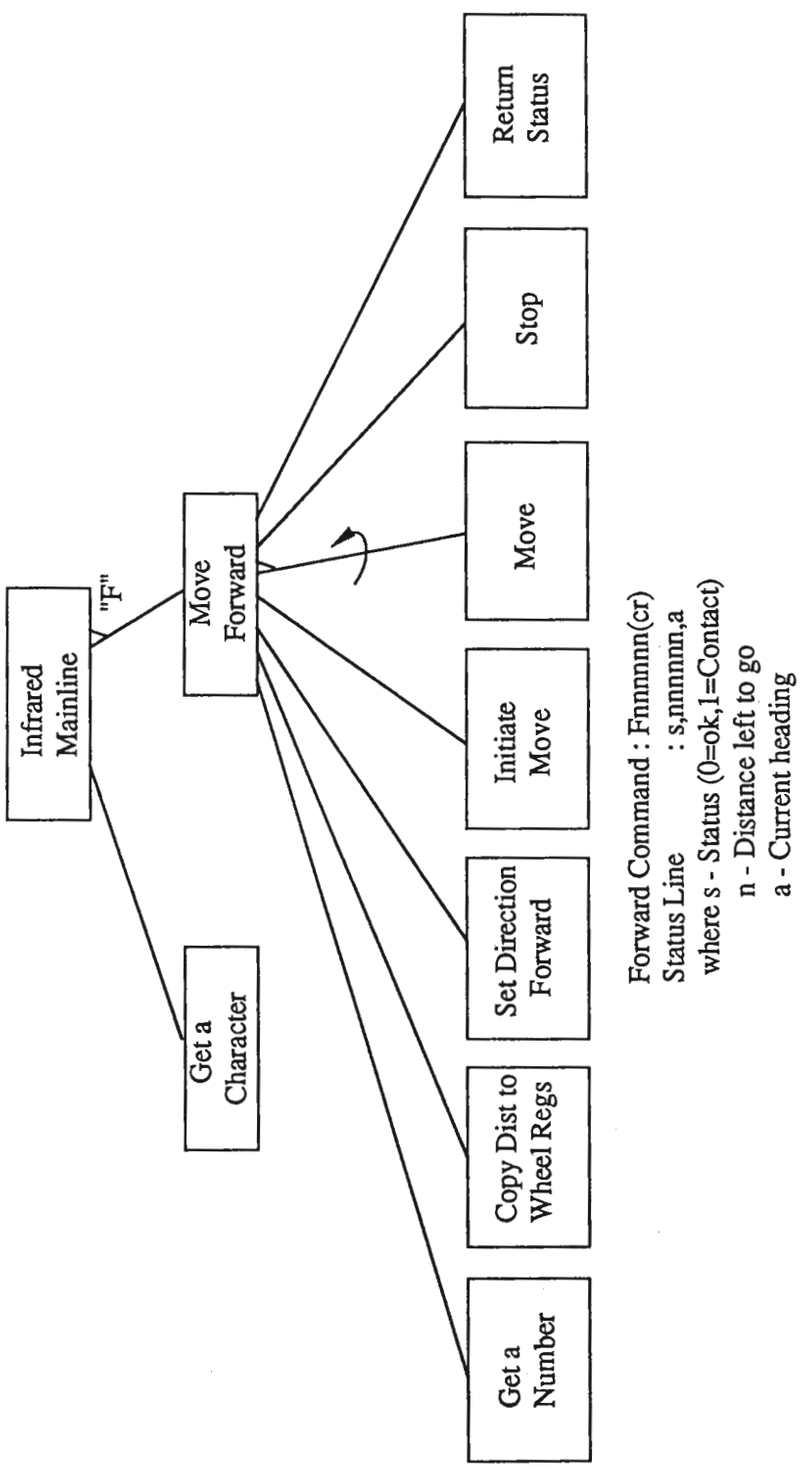


Fig A.2 Structure chart of forward command subroutine

A.3.3 Move Routine

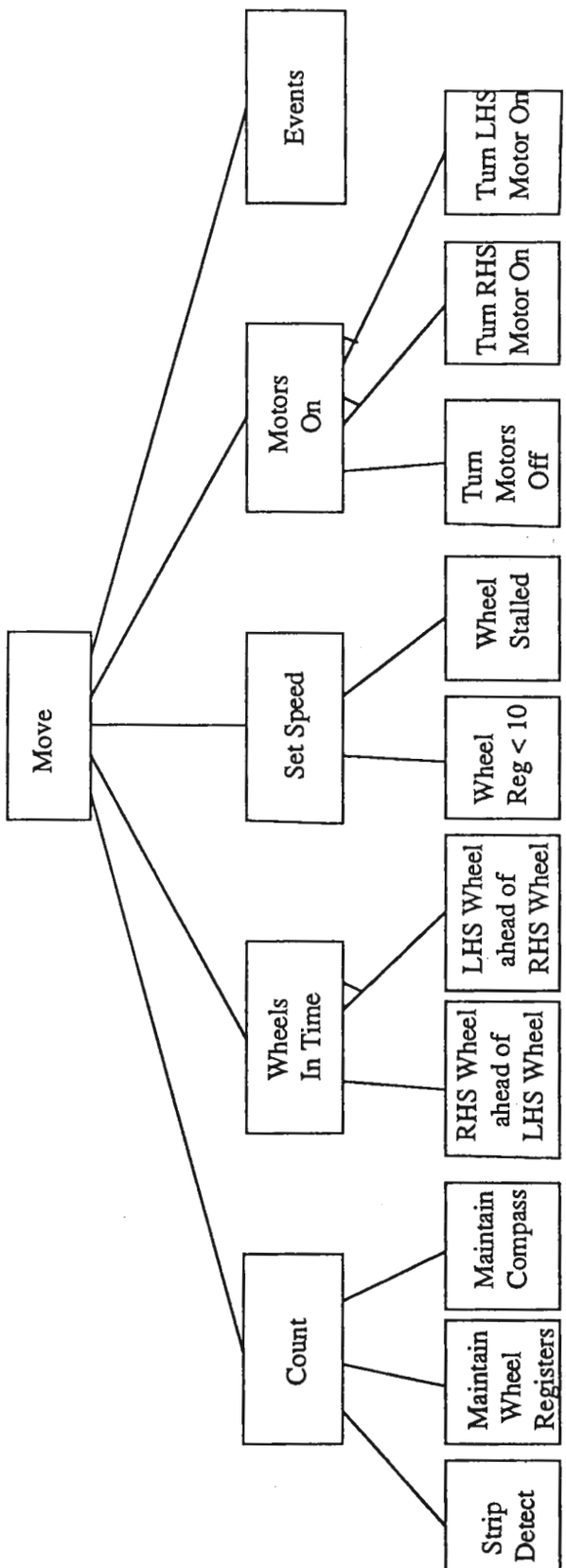


Fig A.3 Structure chart of move subroutine

A.3.4 Event Polling Routine

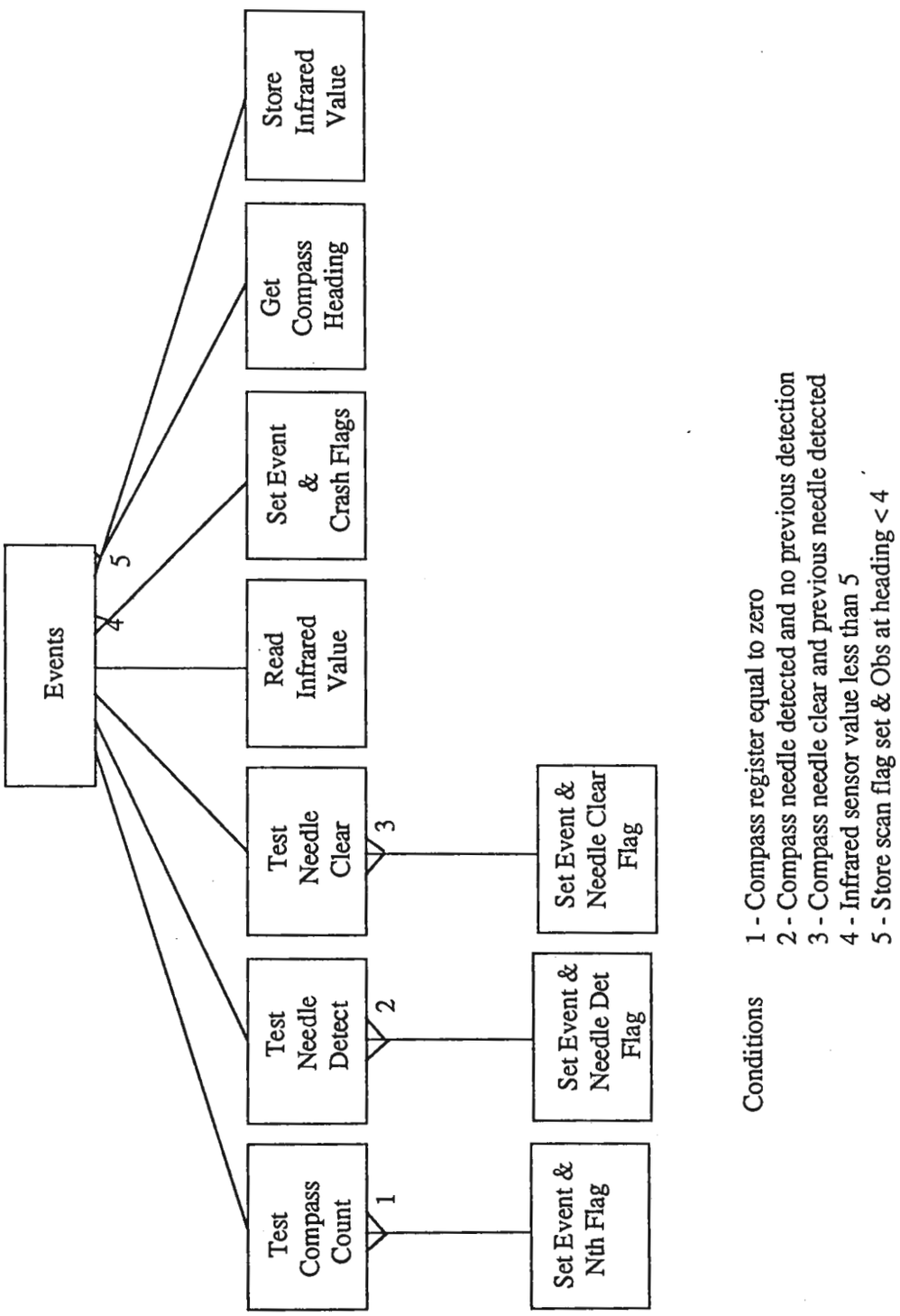


Fig A.4 Structure chart of event polling subroutine

A.4 Micro-Controller Structure Charts

A.4.1 Command Handler

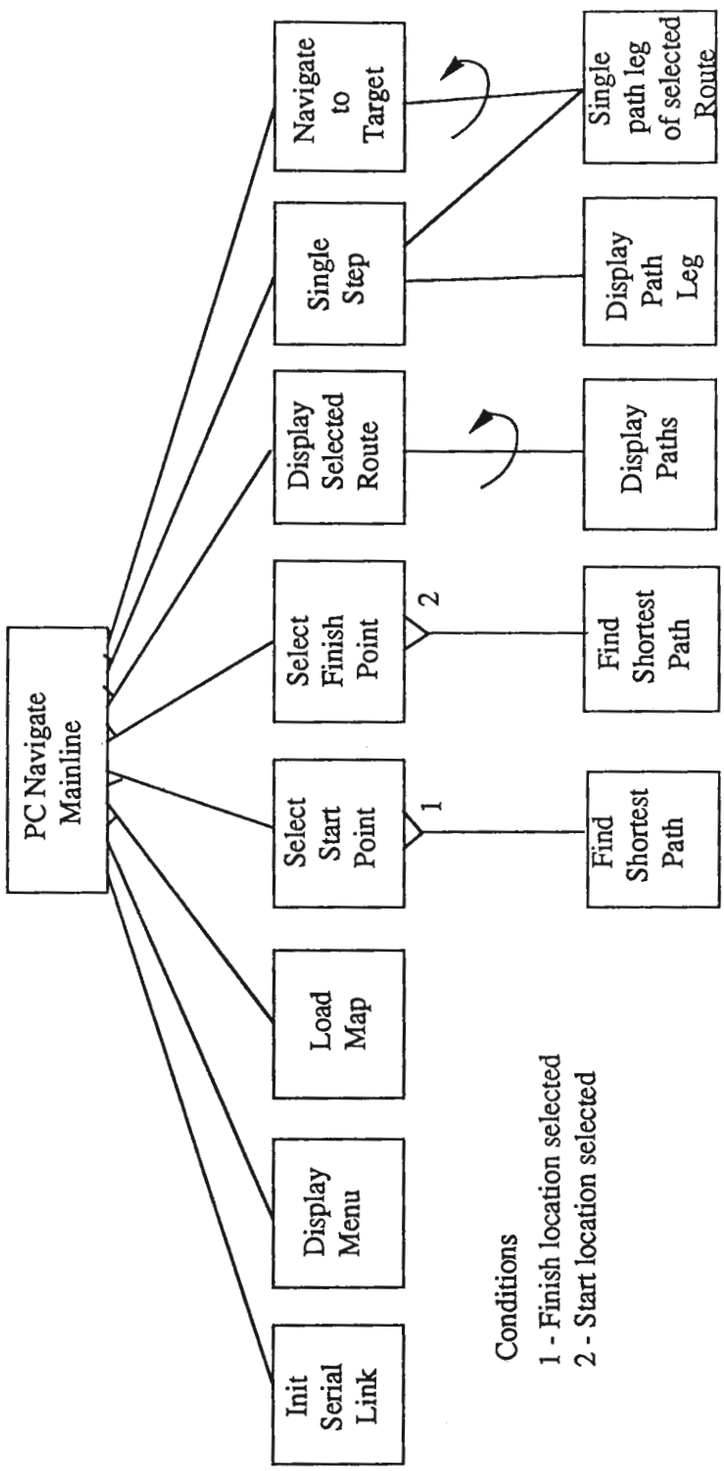


Fig A.5 Structure chart of navigation program

A.4.2 Single Step

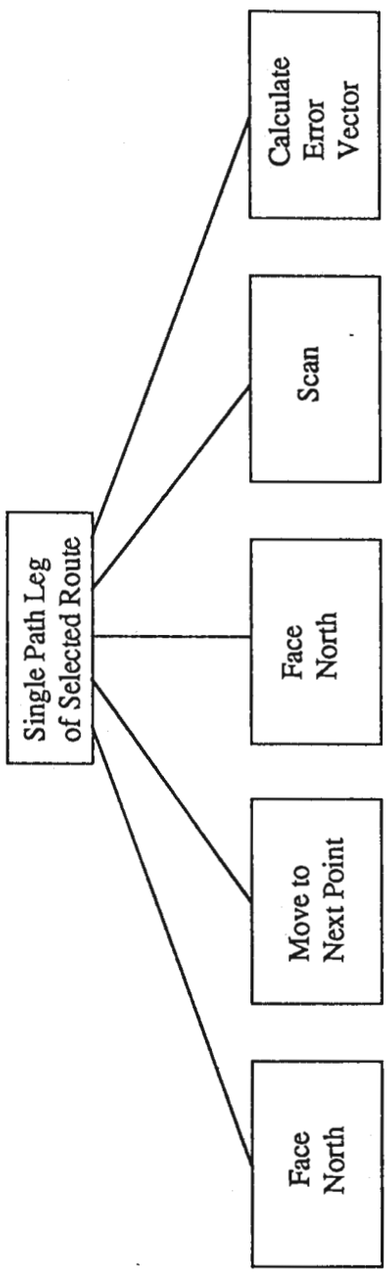


Fig A.6 Structure chart of single step subroutine

A.5 Infra-red Value to Millimetre Conversion Chart

The conversion chart given below is only an example of the kind of Infra-red Signal value to Millimetre curve that can be expected using the infra-red sensor described. By adjusting the sensor the curve moves both in scale and in position up the Infra-red Signal value axis. These results were taken from a conversion table file used when testing the correction capabilities of the navigation software on the 14/Jul/1990.

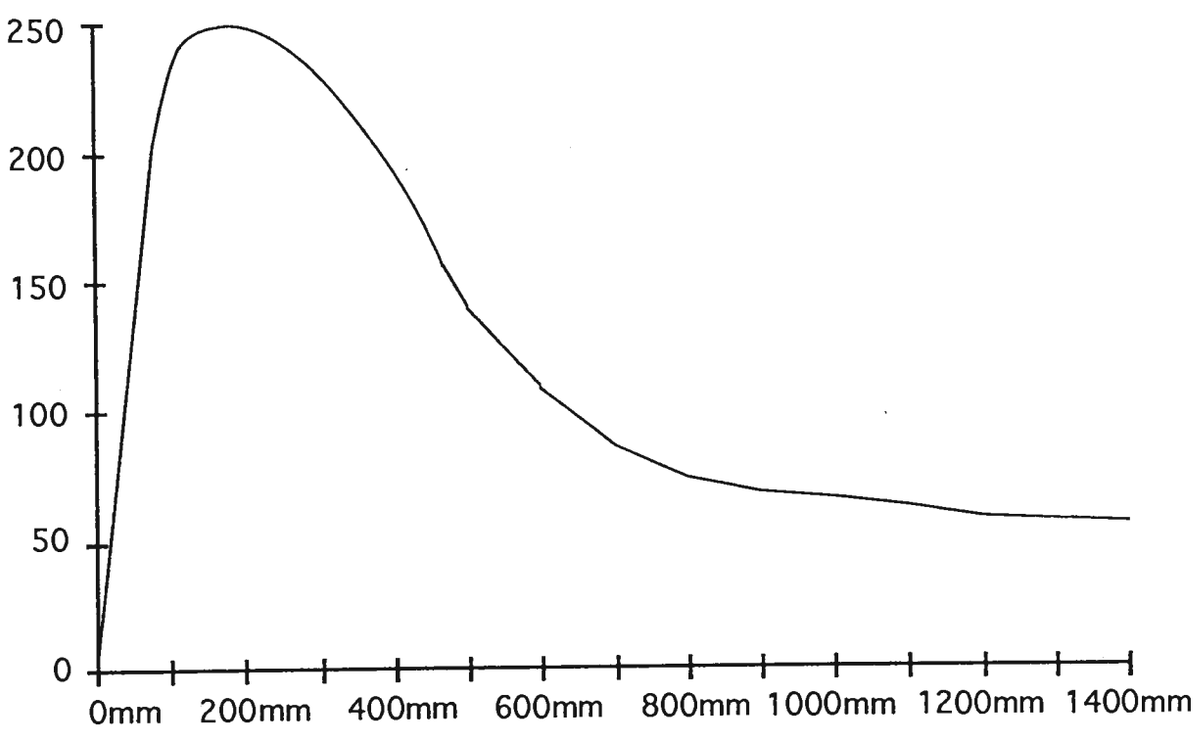
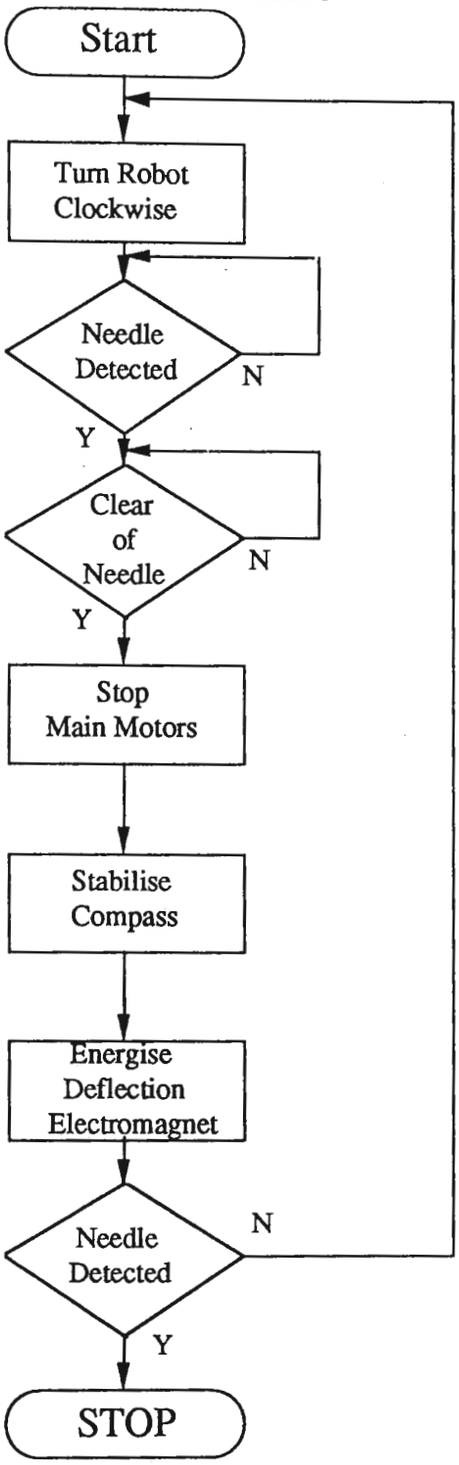


Fig A.7 Infra-red sensor readings vs distance

A.6 Compass Control

A.6.1 North Detection Flow Chart



Note: When robot is pointing north east
needle swings Anti-clockwise
(across sensor).
When robot is point south west needle
swings Clockwise (away from sensor).

Fig A.8 Compass control subroutine

```
FILE: IRCMDS:DAVID      HEWLETT-PACKARD: 8051 Assembler Version 1.06          Sun, 18 Nov 1990, 12:40    PAGE  10

LOCATION OBJECT CODE LINE     SOURCE LINE

                     237 ;
                     238 ;
                     239 ;
                     240             GLOBAL  FACE_NTH
                     241
   010C 120000       242 FACE_NTH:   CALL    GET_CHR             ; READ CR/LF
                     243
   010F E528         244             MOV     A,NTHLSTR           ; MIGHT BE NTH ALREADY ?
   0111 7004         245             JNZ     F_NTH0              ; NO
   0113 E529         246             MOV     A,NTHMSTR
   0115 6000         247             JZ      F_NTH2              ; YES IT IS, NO NEED TO TURN
                     248                                         ; ROBOT AT ALL
                     249
   0117 7A00         250 F_NTH0:     MOV     R2,#0               ; FACE WHAT SHOULD BE NORTH
   0119 7B00         251             MOV     R3,#0
   011B 120412       252             CALL    HEADING
                     253
   011E 120000       254 F_NTH1:     CALL    M_STOP              ; STOP MOTORS AND BRAKE
   0121 120189       255             CALL    CMPSTAB             ; WAIT FOR STABLE COMPAS
                     256
   0124 30B524       257 F_NTH2:     JNB     P3.COMPASB,F_NTH5   ; TURN CLK WISE IF ON TOP OF
                     258                                         ; COMPAS NEEDLE.
                     259
   0127 120171       260             CALL    TIME_ENG            ; TIME TO SWING TO NORTH
   012A 601F         261             JZ      F_NTH5              ; ? MUST BE TO NORTH WEST
                     262
   012C 120197       263             CALL    SWING_TAB           ;  CALC SIZE OF ERROR
                     264
   012F 6035         265             JZ      F_NTH8              ;  LESS THAN 1 LOOP OFF NTH OK
                     266
   0131 F8           267             MOV     R0,A                ;  TURN TO WEST AMOUNT OF ERROR
   0132 7900         268             MOV     R1,#0
   0134 120000       269             CALL    COPY_DIST
                     270
   0137 D208         271             SETB    FWD1                ;  ANTI-CLOCKWISE
   0139 C20A         272             CLR     FWD2
   013B 120000       273             CALL    INIT_MOV            ;  SET DIRECTION RELAYS
                     274
   013E 120000       275 F_NTH3:     CALL    MOVE                ;  PERFORM MOVE
                     276
   0141 B40102       277             CJNE    A,#MOVEBANG,F_NTH4  ;  STOP IF WE HAVE HIT SOMTHING
   0144 801B         278             SJMP    F_NTH7
                     279
   0146 B400F5       280 F_NTH4:     CJNE    A,#MOVEOK,F_NTH3    ;  NOTHING SERIOUS, KEEP GOING
                     281
   0149 80D3         282             SJMP    F_NTH1
                     283
```

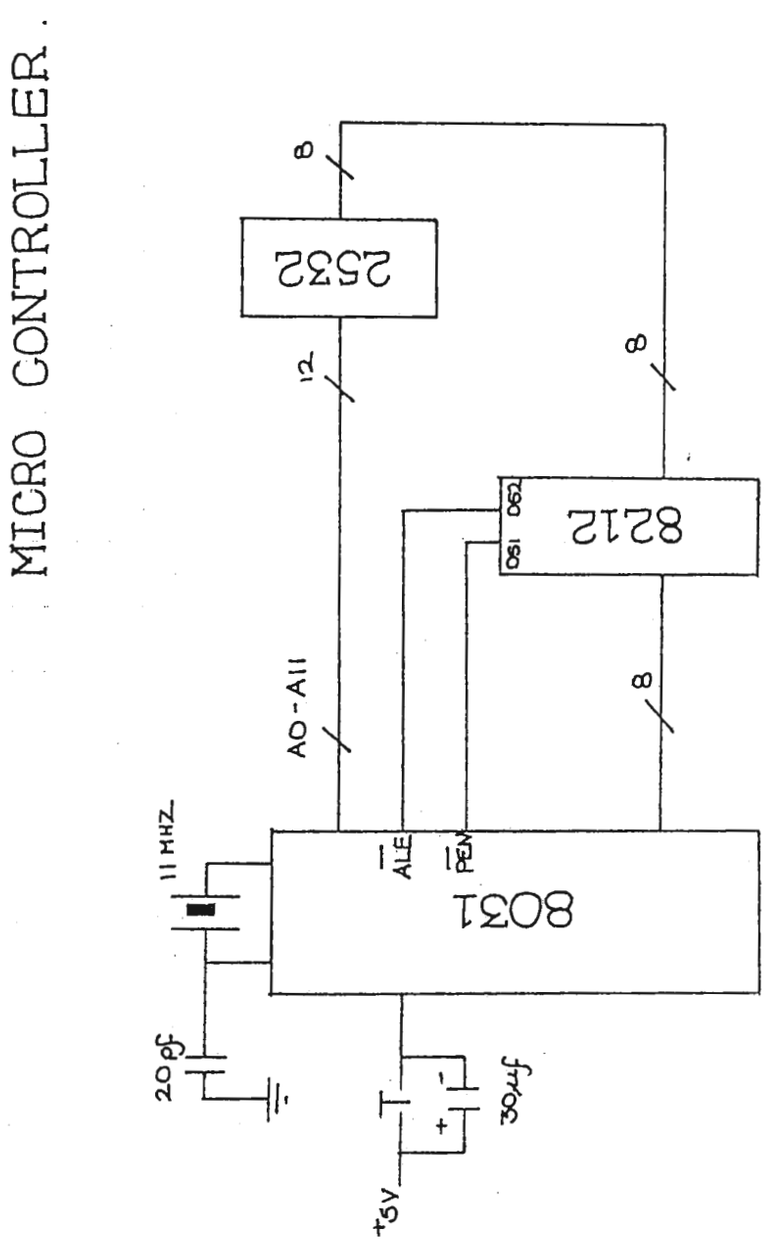
```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     285
                     286 ;-------------------------------------------------;
                     287 ;  WEST OF NORTH SO TURN CLOCKWISE UNTIL COMPAS OK  ;
                     288 ;-------------------------------------------------;
                     289 ;
                     290
     014B C208       291 F_NTH5:    CLR     FWD1                    ; WEST OF NORTH, TURN RIGHT
                     292
     014D D20A       293            SETB    FWD2                    ; SET DIRECTION CLOCKWISE.
     014F 120000     294            CALL    INIT_MOV
                     295
     0152 C291       296            CLR     P1ON1                   ; START MOTORS
     0154 C293       297            CLR     P1ON2
                     298
     0156 30B503     299            JNB     P3.COMPASB,F_NTH6       ; ON TOP OF NEEDLE
     0159 20B5FD     300            JB      P3.COMPASB,$            ; WEST OF NORTH
     015C 30B5FD     301 F_NTH6:    JNB     P3.COMPASB,$            ; ON TOP OF NEEDLE
                     302
     015F 80BD       303            JMP     F_NTH1
                     304
     0161 120000     305 F_NTH7:    CALL    M_STOP          ; BAIL OUT, SOMTHING IS WRONG
     0164 8007       306            SJMP    F_NTH9
                     307
     0166 752800     308 F_NTH8:    MOV     NTHLSTR,#0      ; SET CURRENT DIRECTION TO NTH
     0169 752900     309            MOV     NTHMSTR,#0
     016C E4         310            CLR     A               ; SET STATUS TO OK.
                     311
     016D 120000     312 F_NTH9:    CALL    STATUS
     0170 22         313            RET
                     314
```

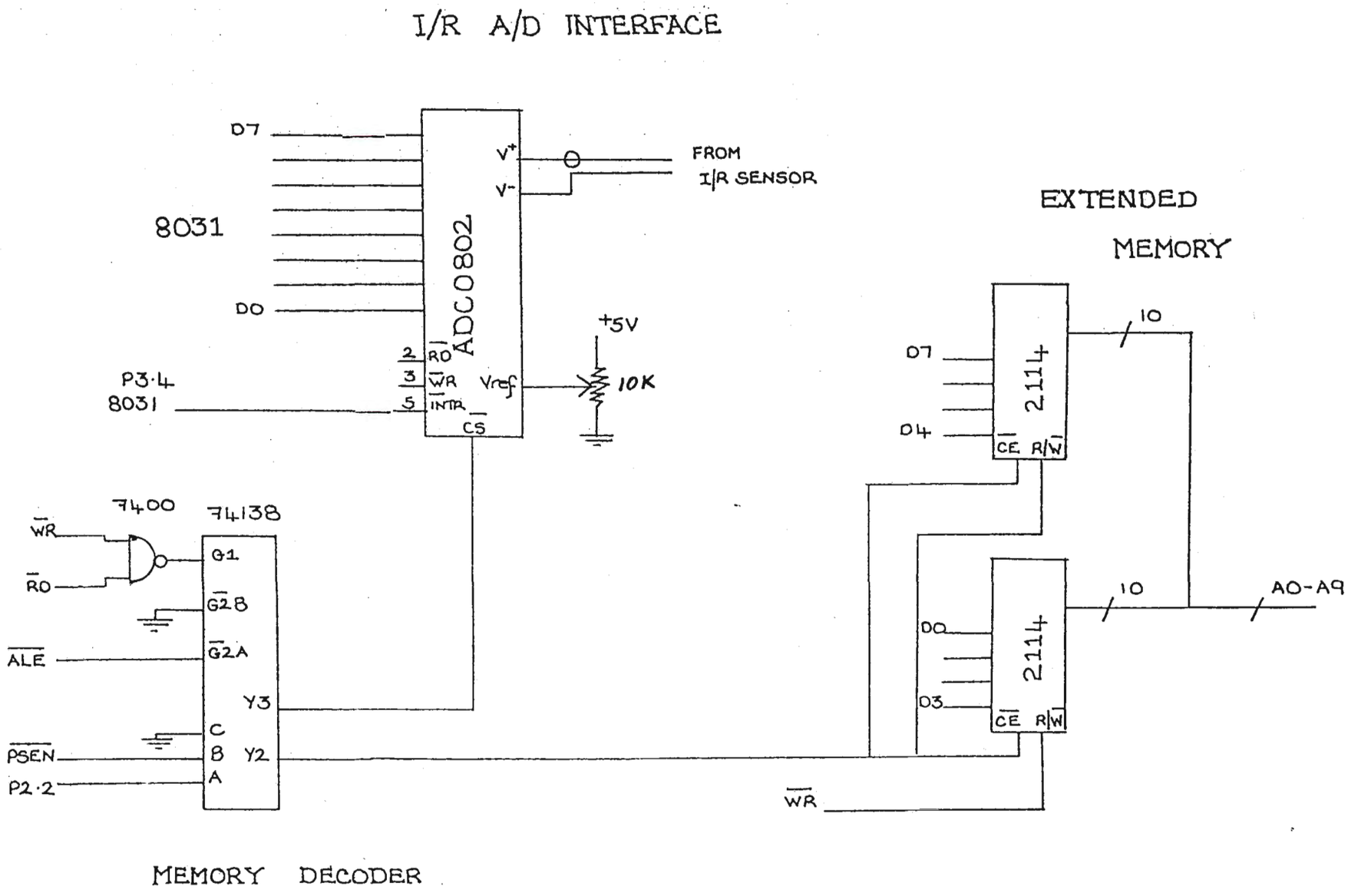
```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     316
                     317 ;-------------------------------------------------;
                     318 ;       ENERGISE COIL AND WAIT FOR COMPAS NEEDLE   ;
                     319 ;-------------------------------------------------;
                     320 ;
                     321
    0171 E51B        322 TIME_ENG: MOV       A,CSWING
    0173 F8          323           MOV       R0,A
    0174 A92C        324           MOV       R1,#MOVDIFF
    0176 7A00        325           MOV       R2,#0
                     326
    0178 D296        327           SETB      P1.MAGENGB
    017A 308508      328 T_ENG1:   JNB       P3.COMPASB,T_ENG2
    017D DAFB        329           DJNZ      R2,T_ENG1
    017F D9F9        330           DJNZ      R1,T_ENG1
    0181 A92C        331           MOV       R1,#MOVDIFF
    0183 D8F5        332           DJNZ      R0,T_ENG1
                     333
    0185 C296        334 T_ENG2:   CLR       P1.MAGENGB
    0187 E8          335           MOV       A,R0
    0188 22          336           RET
                     337
                     338
                     339 ;-------------------------------------------------;
                     340 ;         WAIT FOR COMPAS NEEDLE TO SETTLE.       ;
                     341 ;-------------------------------------------------;
                     342 ;
                     343
    0189 E51C        344 CMPSTAB:    MOV       A,CSTAB
    018B F8          345             MOV       R0,A
    018C 7900        346             MOV       R1,#0
    018E 7A00        347             MOV       R2,#0
                     348
    0190 DAFE        349 C_STAB1:    DJNZ      R2,C_STAB1
    0192 D9FC        350             DJNZ      R1,C_STAB1
    0194 D8FA        351             DJNZ      R0,C_STAB1
                     352
    0196 22          353             RET
                     354
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     356 ;-------------------------------------------------;
                     357 ;         LOOKUP TABLE TO FINDOUT HOW FAR TO TURN ;
                     358 ;         ANTICLOCKWISE TO BE RIGHT ON NORTH.     ;
                     359 ;-------------------------------------------------;
                     360 ;
                     361
    0197 04          362 SWING_TAB: INC     A
    0198 83          363            MOVC    A,@A+PC
    0199 22          364            RET
                     365
    019A 2C          366 TAB1:      DB      44          ; 0
    019B 2B          367            DB      43          ; 1
    019C 2A          368            DB      42          ; 2
    019D 29          369            DB      41          ; 3
    019E 28          370            DB      40          ; 4
    019F 27          371            DB      39          ; 5
    01A0 26          372            DB      38          ; 6
    01A1 25          373            DB      37          ; 7
    01A2 24          374            DB      36          ; 8
    01A3 23          375            DB      35          ; 9
    01A4 22          376            DB      34          ; 10
    01A5 21          377            DB      33          ; 11
    01A6 20          378            DB      32          ; 12
    01A7 1F          379            DB      31          ; 13
    01A8 1D          380            DB      29          ; 14
    01A9 1C          381            DB      28          ; 15
    01AA 1A          382            DB      26          ; 16
    01AB 19          383            DB      25          ; 17
    01AC 17          384            DB      23          ; 18
    01AD 16          385            DB      22          ; 19
    01AE 14          386            DB      20          ; 20
    01AF 13          387            DB      19          ; 21
    01B0 11          388            DB      17          ; 22
    01B1 10          389            DB      16          ; 23
    01B2 0F          390            DB      15          ; 24
    01B3 0F          391            DB      15          ; 25
    01B4 0E          392            DB      14          ; 26
    01B5 0D          393            DB      13          ; 27
    01B6 0D          394            DB      13          ; 28
    01B7 0C          395            DB      12          ; 29
    01B8 0B          396            DB      11          ; 30
    01B9 0B          397            DB      11          ; 31
    01BA 0A          398            DB      10          ; 32
    01BB 09          399            DB      09          ; 33
    01BC 08          400            DB      08          ; 34
    01BD 07          401            DB      07          ; 35
    01BE 06          402            DB      06          ; 36
    01BF 05          403            DB      05          ; 37
    01C0 05          404            DB      05          ; 38
    01C1 04          405            DB      04          ; 39
    01C2 04          406            DB      04          ; 40
    01C3 04          407            DB      04          ; 41
    01C4 03          408            DB      03          ; 42
    01C5 03          409            DB      03          ; 43
    01C6 02          410            DB      02          ; 44
    01C7 02          411            DB      02          ; 45
```

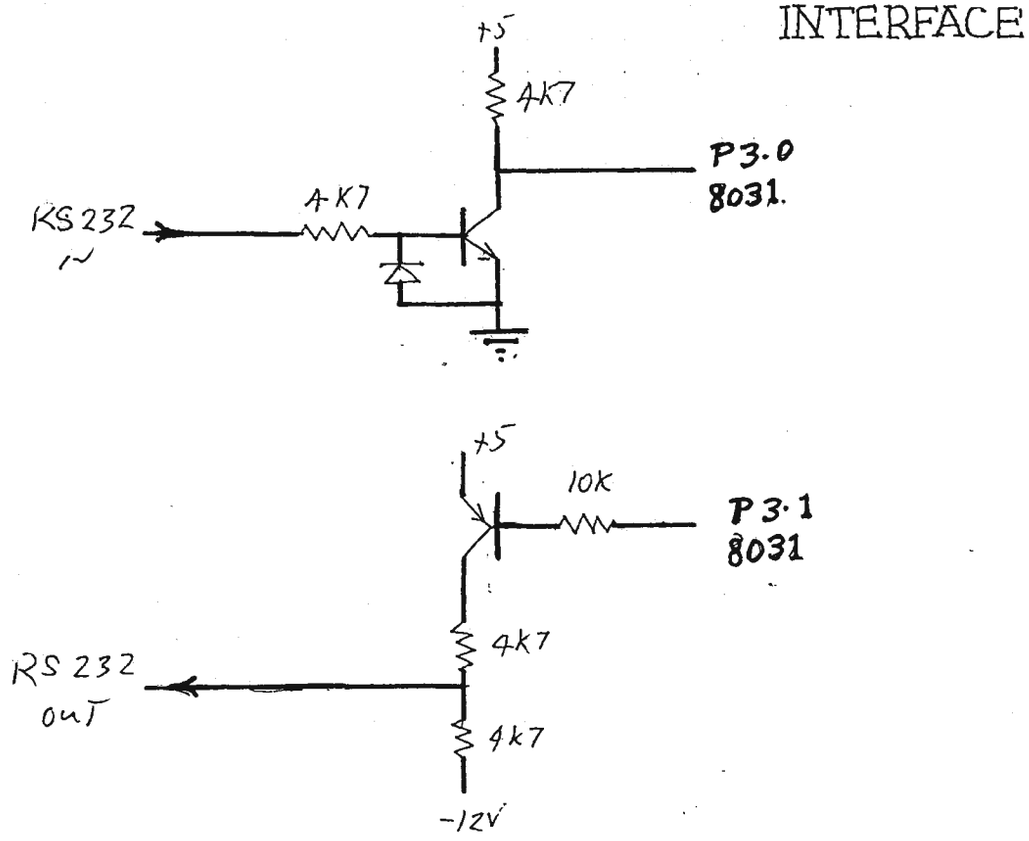
```
LOCATION OBJECT CODE LINE      SOURCE LINE

    01C9 01          413          DB      01          ; 47
    01CA 00          414          DB      00          ; 48
    01CB 00          415          DB      0           ; 49
    01CC 00          416          DB      0           ; 50
    01CD 00          417          DB      0           ; 51
    01CE 00          418          DB      0           ; 52
    01CF 00          419          DB      0           ; 53
    01D0 00          420          DB      0           ; 54
    01D1 00          421          DB      0           ; 55
    01D2 00          422          DB      0           ; 56
    01D3 00          423          DB      0           ; 57
    01D4 00          424          DB      0           ; 58
    01D5 00          425          DB      0           ; 59
    01D6 00          426          DB      0           ; 60
                     427
```

A.7 Circuit Diagrams

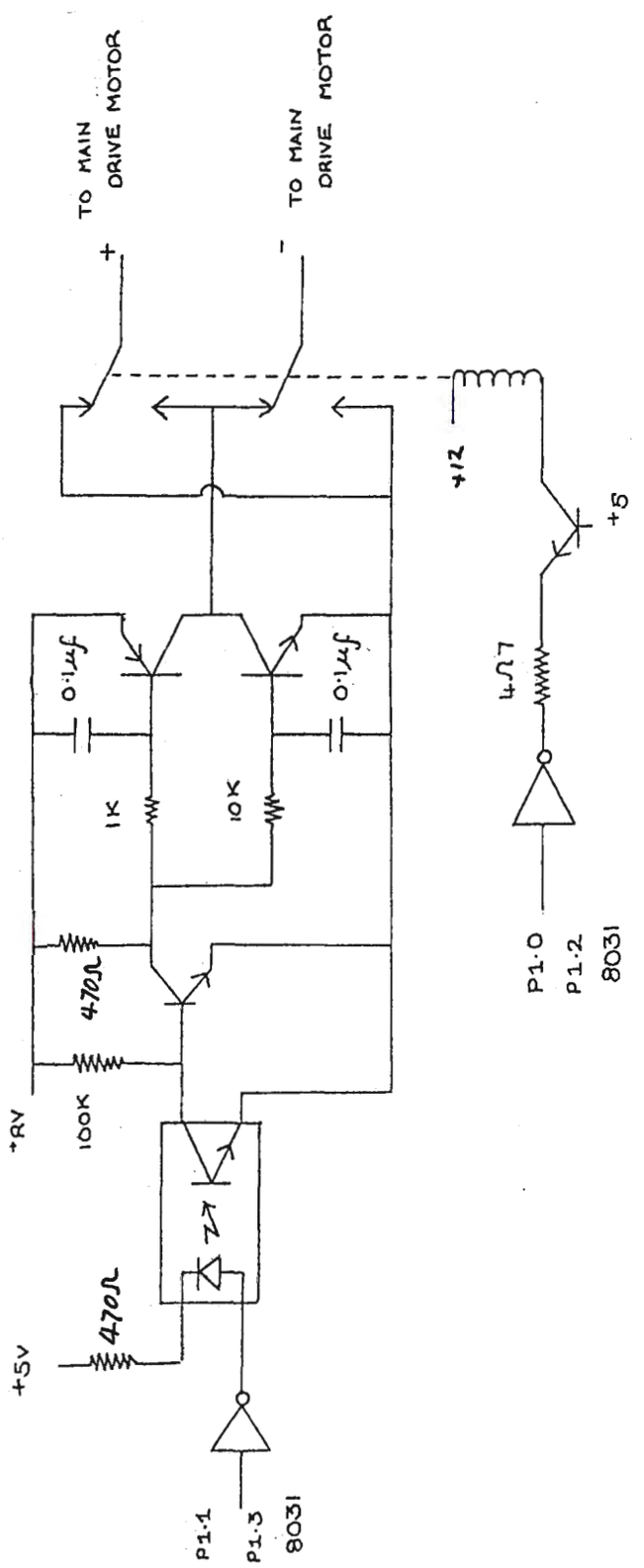



I/R A/D INTERFACE
8031
D7
D0
ADC0802
V+
V-
FROM
I/R SENSOR
+5V
10K
Vref
RD
WR
INTR
CS
2
3
5
P3.4
8031
EXTENDED
MEMORY
2114
D7
D4
CE
R/W
10
D0
D3
AO-A9
7400
74138
WR
RD
G1
G2B
G2A
ALE
Y3
C
B
A
Y2
PSEN
P2.2
MEMORY DECODER

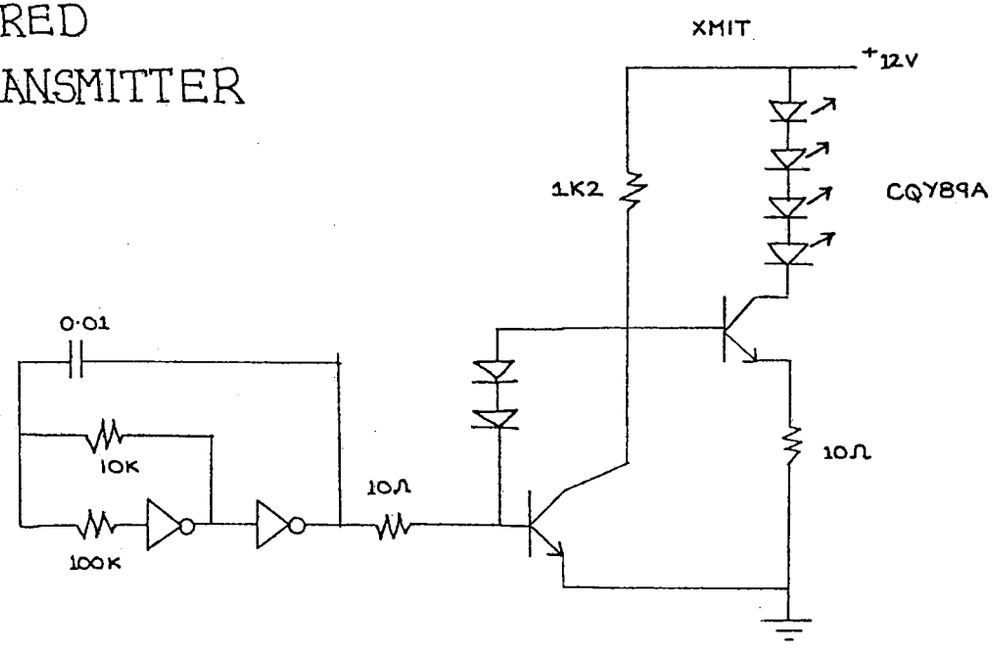
SERIAL INTERFACE



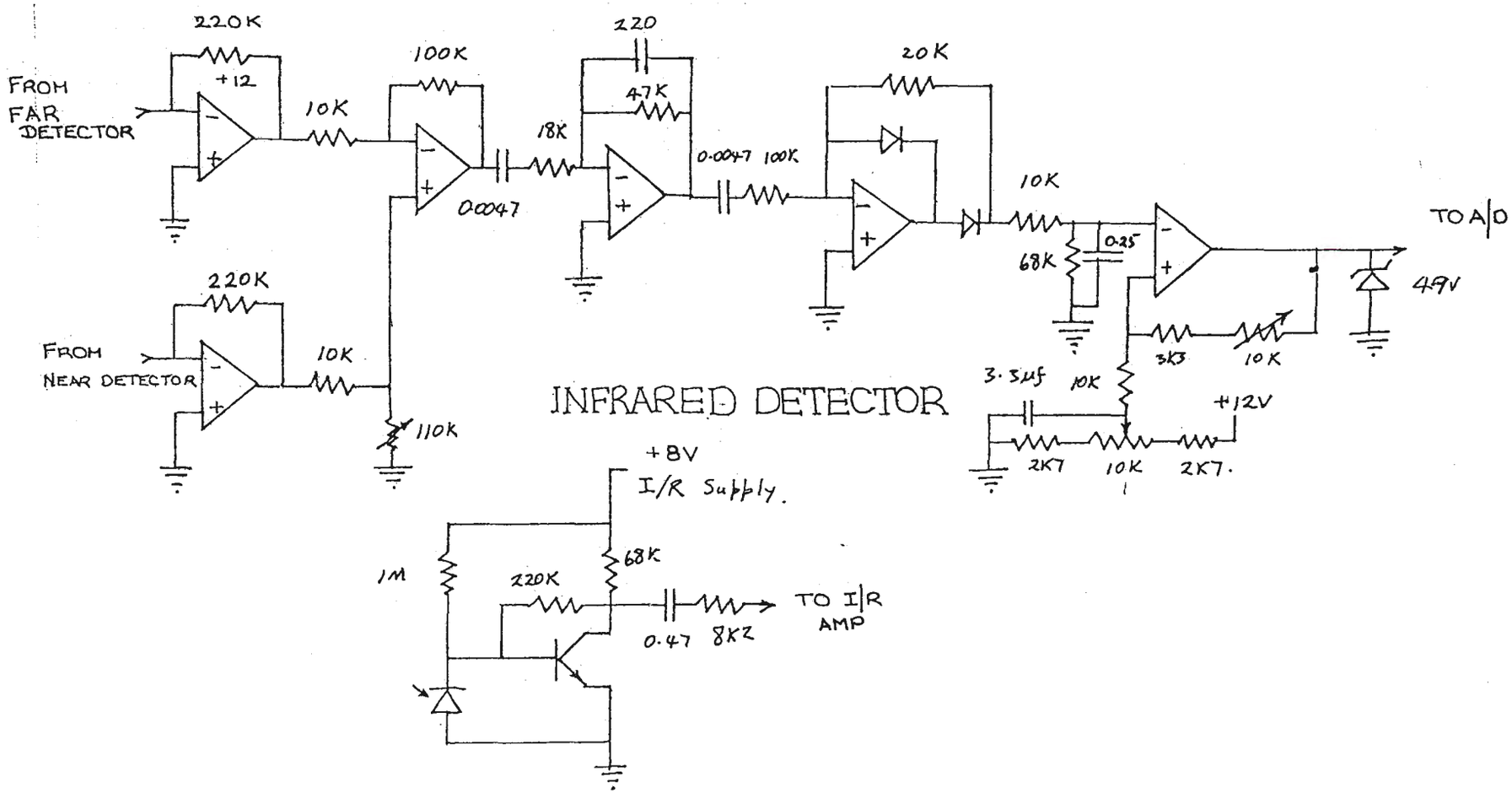
DRIVE MOTOR CONTROL



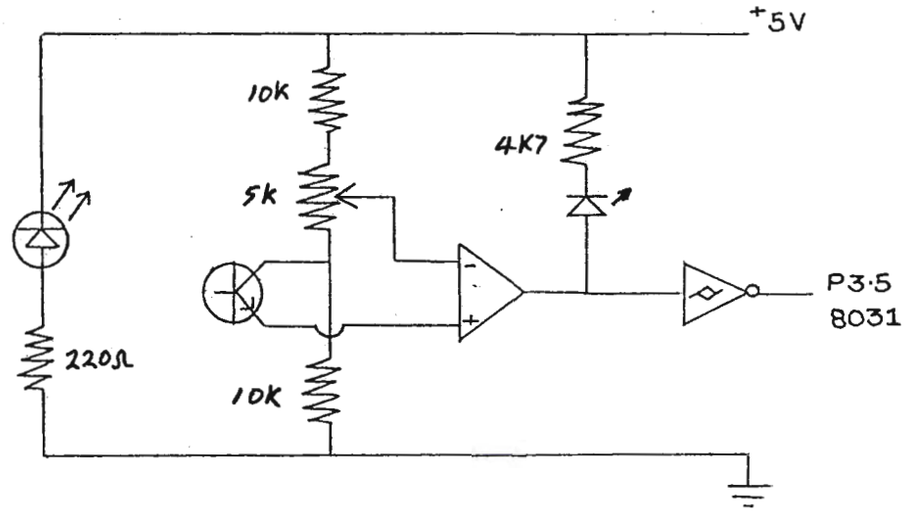
INFRARED TRANSMITTER



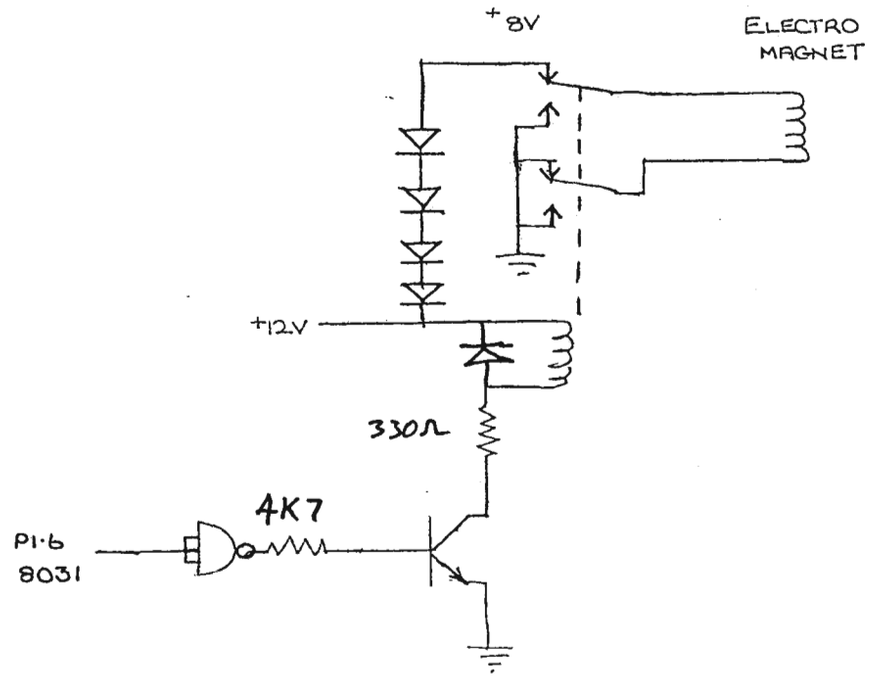
INFRARED SIGNAL PROCESSOR



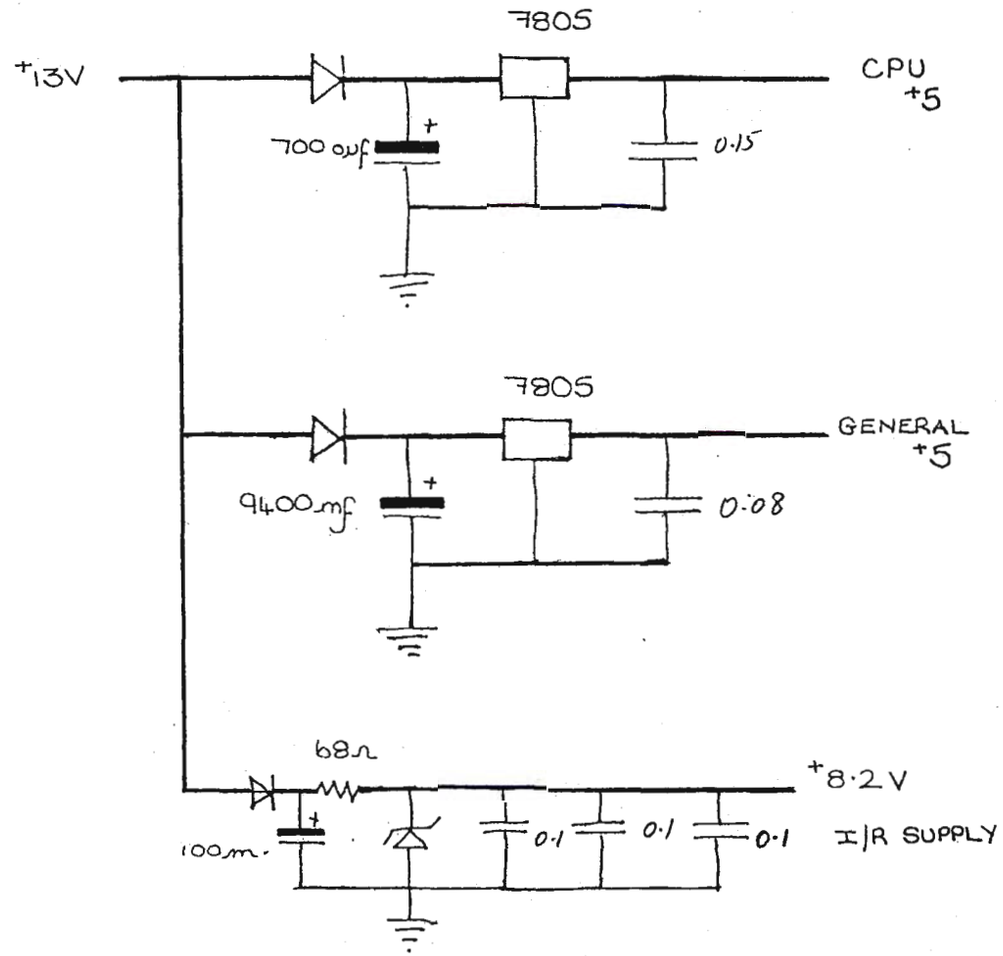
COMPASS NEEDLE DETECTOR



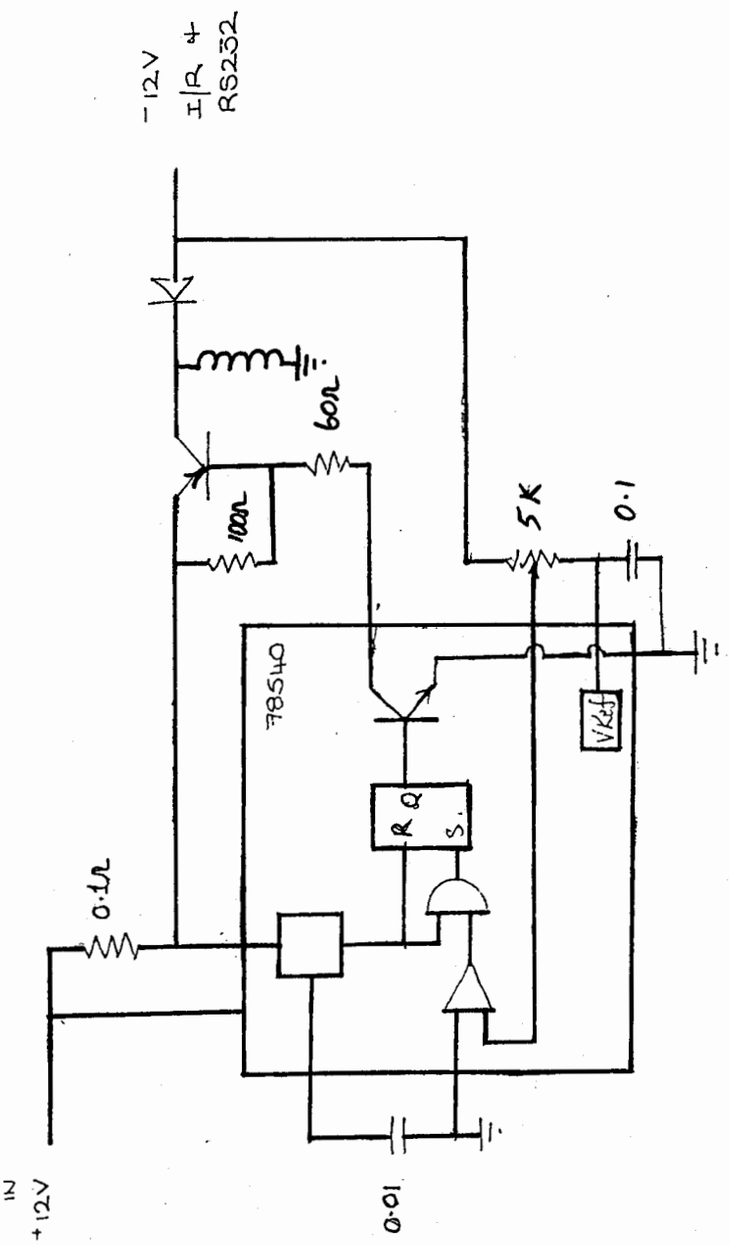
COMPASS DEFLECTION



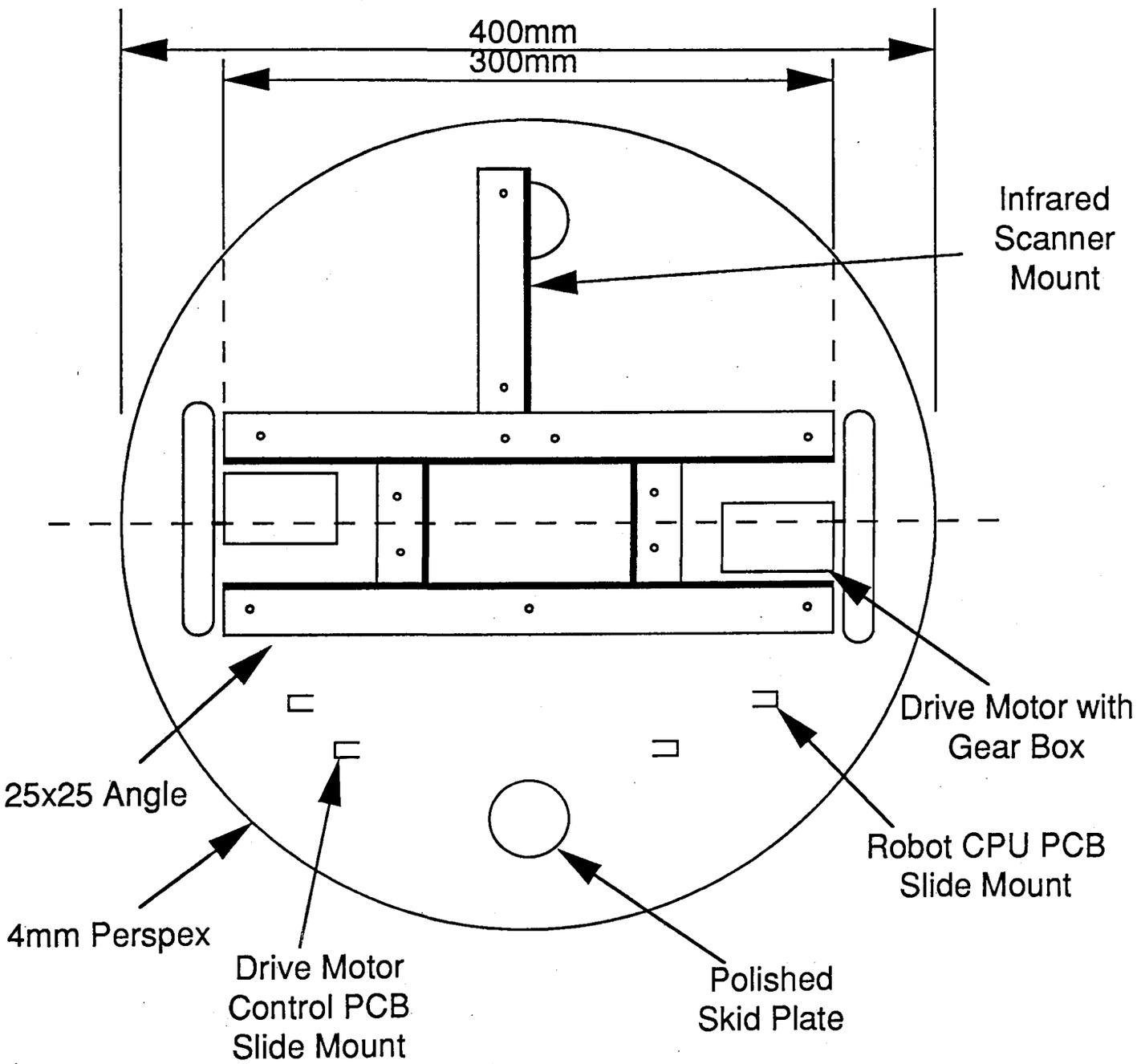
POWER REGULATOR



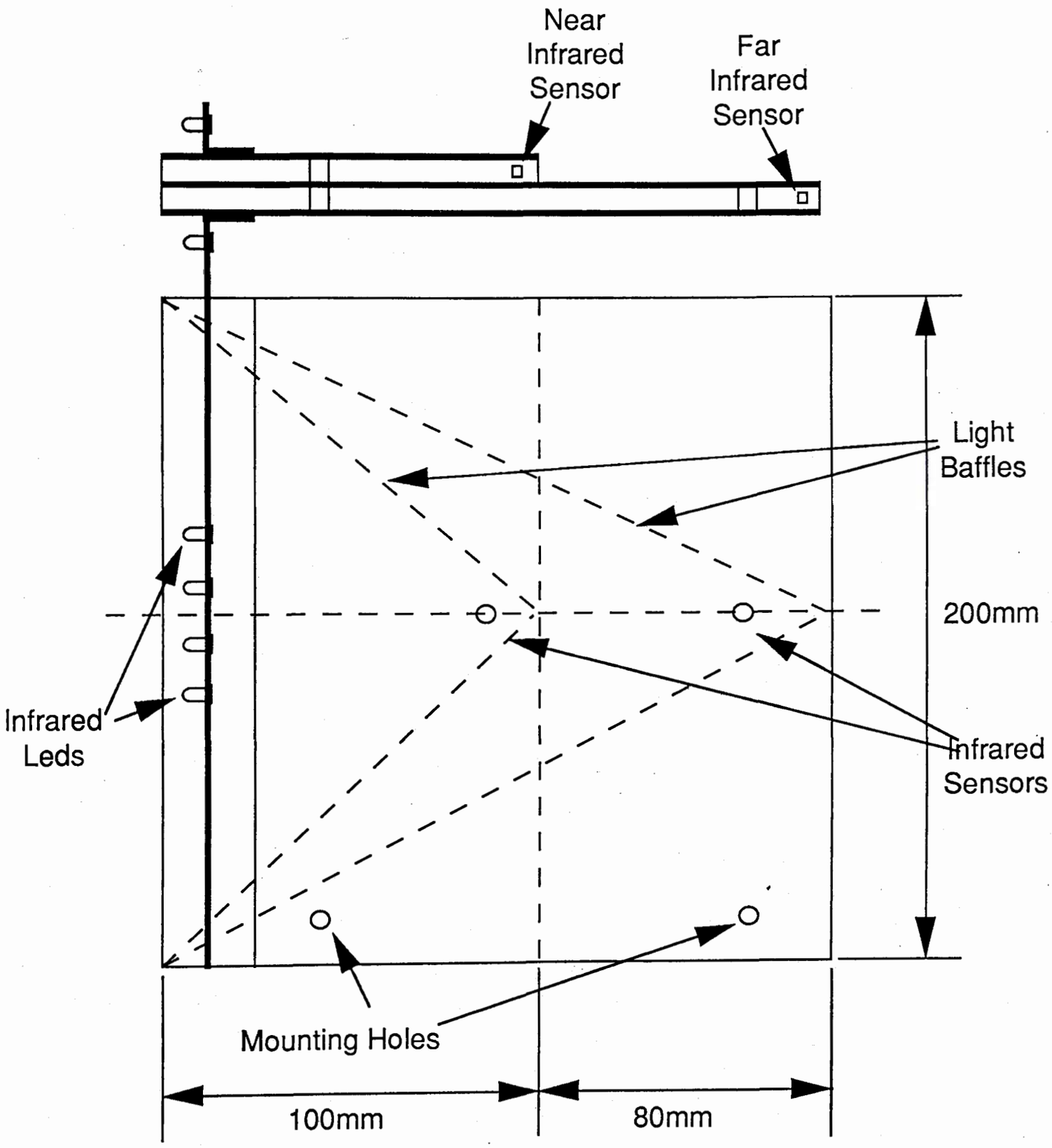
POWER REGULATOR



A.8 Configuration Diagrams



Robot Base Plan



Infrared Distance Sensor