Localization of Mobile Robots Using Magnetic Fields

Cory McKay
Department of Physics
McGill University
3600 University Street
Montreal, QC, Canada, H3A 2T8

The Minh Luong
Department of Physics
McGill University
3600 University Street
Montreal, QC, Canada, H3A 2T8

Abstract

Methods that robots could use to automatically determine their position in different environments were explored. Experiments were conducted using a compass, where our robot constructed maps of the magnetic fields in environments and later used them to find its position. Some discussion was also made of localization methods that use sonar. Our results showed that our robot had little trouble determining its position in environments that include a reasonable amount of variation in magnetic fields, such as buildings containing metal structures.

1) Introduction

Mobile robots are extremely useful in a diverse range of fields. A vast array of equipment can be mounted on them, enabling them to perform innumerable tasks that would be difficult for humans to do and allowing them to perform precision operations. They are also extremely useful in locations which are hostile to human life, such as radiated areas, or clean environments which human workers could contaminate. By automating these robots, as opposed to operating them directly from a remote location, the robots can perform tasks more precisely than they could under direct human supervision and they can easily perform extremely repetitive tasks that could cause a human operator to become bored and careless.

One of the central issues in implementing this automation is enabling the robot to independently determine its position. A simplistic approach is to use stepper motors on the robot. Knowing the radius of the robot's wheels, it's position relative to some starting point can easily be found by measuring the number of revolutions that the wheels have made. However, errors can crop up in the stepper measurements, rendering them essentially meaningless after a certain number of movements have been made. This is particularly true if the movement of the robot involves many starts and stops or if the movements are very short. Despite these limitations, many companies and research institutions still use this method. They have a technician periodically recalibrate the robot manually so that the stepper error does not have the opportunity to accumulate. However, it would be much preferable to eliminate this human interaction and have the robot perform autonomously.

Research has been done into many different methods of automated position determination. Which method is the best depends on the type of environment that the robot will be operating in and what types of jobs it will be performing. For example, one method is place geometrical patterns on either the robot or in the environment and have the robot use a camera to extrapolate its position relative to these patterns. This is a very useful method in static environments, but its limitations become apparent in environments where there are moving objects which can block the field of view from the camera to the pattern. This kind of limited utility is typical of most of the orientation methods which have been developed to date and one needs to select the method that is best suited to one's particular needs.

Many of the successful methods do use the stepper measurements discussed earlier, but they periodically measure some other characteristic of the environment, and use this information to automatically recalibrate the stepper readings so that stepper error is eliminated before it has the chance to accumulate to significant levels. To do this, the robot must first construct a map of the environment that it will be operating in with regards to the characteristic that is being measured. Again, what this characteristic is depends on the particulars of the environment. This map is then kept in memory, and the robot can use it to find its position. A major advantage of this method is that it allows the robot to easily and automatically operate in new environments, since it can simply construct a new map on demand. It was thus our goal in this lab to produce a method whereby a mobile robots can be placed in an environment, automatically map it out, and then use this map to determine its location reliably and easily. Ideally, it should be possible to place the robot in a random location in the environment and have it be able to tell where it is .

The method that we focused on involved constructing maps of magnetic fields. There is a certain amount of variation in the magnetic field of the earth, and we had our robot use this variation to distinguish between different points in an environment. The variation is quite small, so a very sensitive compass is needed to detect it. However, the variation can become relatively large in regions that contain a large amount of metal. Since the support structures of most buildings contain a good deal of metal, there should be more than enough spatial variance for a decent compass to be able to distinguish between different points. Our goal in this experiment was to discover if there was enough variation in the magnetic fields of such an environment for the robot to be able to determine its location successfully, and to devise a method by which this could be done.

We also looked into localization methods involving sonar measurements. However, this is really an entirely different project and time constraints prevented us from writing code to actually implement the method we devised and seeing if it works. An outline of it will still be included in this report, though, since the method could be quite useful and it would be an interesting area for future experimenters to explore.

2) Experimental Methods, Results and Discussion

2.1) Environment

Our experiment was conducted in the Mobile Robotics Lab in the McGill Center for Intelligent Machines. We were not permitted to take the robot outside of this environment, which was unfortunate, as we would have liked to see how our orientation method worked in other buildings or outside. Even so, there is a large testing space in the Mobile Robotics Lab, so we had no trouble performing our experiment there. We found that there was a great deal of spatial variation in the magnetic field near the walls of the room and near computers, and much less variation in the center of the room. This enabled us to check our procedure in both types of environments. Large plastic sheets were available for us to simulate walls, so that we could divide the testing area into sub-areas.

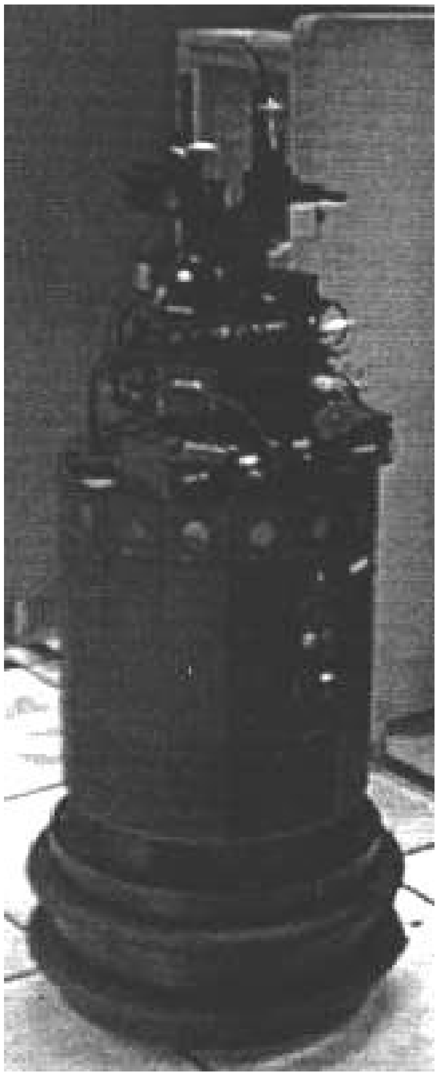

Figure 1: The Nomad 200 robot.

2.2) Equipment

A Nomad 200 robot was used, of the type shown in figure 1. It is able to move and rotate with a high degree of accuracy, and it can measure the number of rotations that its wheels make so as to provide stepper motor data. It attempts to keep track of stepper error as well, which is a very useful feature. For example, if the robot is instructed to move forwards 100.00 centimeters, it might actually only move 100.82 centimeters, and the position values returned by the robot reflect this. Although there is some stepper error which the robot does not account for, these indications helped to provide an indication of what order of error to expect. It is also equipped with several sensory devices, including a compass, sonar (which takes readings in 360 degrees simultaneously), laser range-finder and video camera. We were able to communicate with the robot from our UNIX workstations using a radio modem.

The compass that was used is a digital C100 Compass Engine. It is a microprocessor-controlled fluxgate compass subsystem which consists of a fluxgate sensing element and an electronics board that allows it to be accessed by external software. It interfaces to the robot through a serial port. This was a major source of difficulty for us, as the software used to control the robot is not compatible with the compass, and low-level code had to be written to acquire readings from the serial port (using the PERL language). This was very inconvenient, as we had to write all of our code to communicate simultaneously with two entirely different interface systems. Once this difficulty was overcome, however, the compass worked very well. The biggest problem was that it takes some time for the computer to access the compass, meaning that it was very time consuming to make maps of large areas. The efficiency of the method we used here could be greatly increased if a compass that was designed to interact more naturally with the MRL's controller software could be found, or if the controller software were rewritten to accommodate the compass. In any event, the compass that we used is fine if the robot only needs to operate in a small environment, or if time constraints are not an important issue.

According to the manual, the compass has a minimum error of 0.5 degrees, but we found that it is considerably more accurate than this, at least in terms of random error. By taking readings at the same position at different times, and by moving away from the position and then returning to it, we found that the compass consistently returned the same readings, with a variance of only 0.1 or 0.2 degrees once time variance had been averaged away (time variance is discussed in section 2.4). The measured variations in the magnetic field were consistently higher than this, so compass accuracy was not a problem in this experiment. It is quite possible that the 0.5 degrees that the manual refers to is systematic, but such an error is not relevant to our experiment, since we only cared about variances from point to point. It is of little importance to us if all of our readings are off by 0.5 degrees in the same direction, since the variance from point to point remains the same.

The compass returns a value from 0.0 degrees to 360.0 degrees. It would have been better if the compass ended at 359.9 degrees or started at 0.1 degrees, but it was a simple matter for us to write code to transform readings of 0.0 degrees to 360.0 degrees to avoid errors. The compass also returns a reading of 800 degrees if it notices too much rapid variation in the magnetic field. This was quite useful in helping us to quickly recognize bad data points. The compass was unable to measure the strength of the field. It would be nice if a compass could be found which measured this as well as the direction of the field, since this would give the robot more information which it could use to find its position, but the information about the direction alone is still sufficient.

It takes about four seconds for the reading of the compass to stabilize after being moved. For this reason, we set up all of our programs to wait five seconds after each movement before taking data from the compass.

It was important that the compass be placed directly on the robot's axis of rotation. If this was not done, the compass would have been spatially displaced when the robot rotated, leading to readings at any one location that would not have been indicative of the actual location that was expected. The compass also had to be placed outside the robot to prevent its readings from being shielded from the room's field by the metal casing of the robot.

2.3) Programming issues

A program called Robodaemon was used to interface with the robot. It allowed us to give the robot instructions to move and it returned the stepper position measurements of the robot. It tracked the distance moved and the change in orientation relative to its starting point. It also allows the user to use some of the robot's devices, such as its laser range finder and sonar. However, as mentioned earlier, the compass is incompatible with Robodaemon, and we had to use a separate program to communicate with it.

Robodaemon also has a feature which allows it to create a simulated environment. This

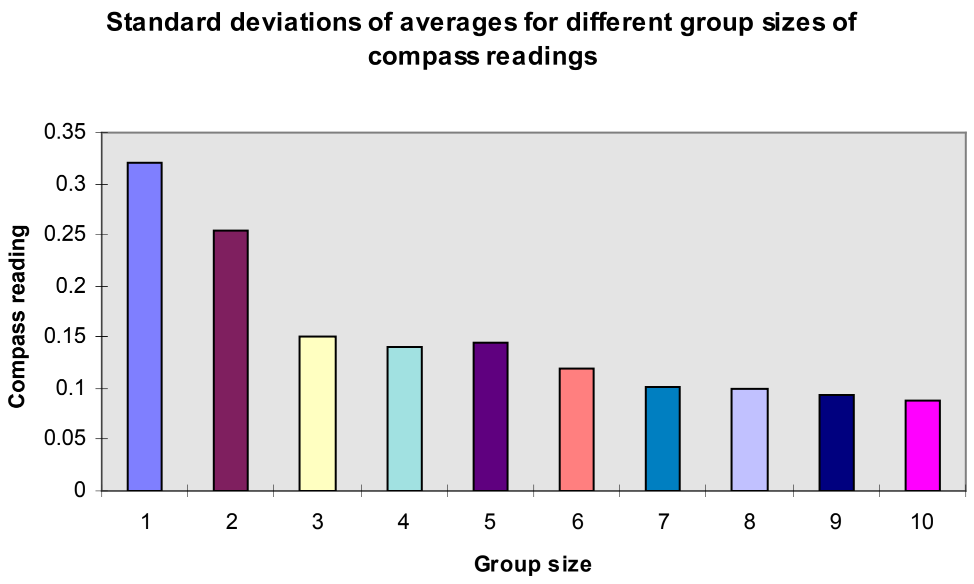

Figure 2: The standard deviations of the averages of compass readings of different group sizes. This was done to eliminate the effects of time variance. The standard deviation was 0.1 degrees with group sizes of seven or more, which was acceptable for our purposes.

was extremely useful in allowing us to debug our programs, since the Nomad 200 was often under repair and unavailable to us, and it also has batteries which can only operate for a few hours before needing to be recharged.

C programs were used to automate the robot's mapping and localization procedures. The first programs that we wrote was MAP, which causes the robot to automatically construct a map of any rectangular environment. The user can specify the dimensions of the environment, as well as the desired density of the map (i.e. the distance between the points that MAP takes readings at) and the number of readings and orientations at which the robot takes data at each point. The other two programs that we wrote, STATIONARY and ANALYZE, respectively allow the robot to take readings at its current position and to compare these readings to its map and extrapolate its position from them. MAP and ANALYZE can be found in Appendix 1. The details of how they work will be discussed in sections 2.5 and 2.6. The first two programs made use of programming libraries to interface with Robodaemon and our PERL program, and were thus able to pass instructions directly to the robot and record readings from it.

2.4) Preliminary measurements

We noticed early on that multiple compass readings at a single point fluctuated as much as 3 degrees at some places in the room, yet were completely stable at other locations. The time-variance was particularly large in areas where there were a lot of electronics. Since the time variance in the worst areas was strong enough to mask the spatial variance that we were measuring, it was necessary to deal with it. To do this, we positioned our robot at the position in the room that showed the most time-variance and had it take many readings at each position. We then divided these readings into groups of different sizes and found the average of each group. We then took groups of the same size and found the standard deviation of their averages. Our data is shown in figure 2.

As one would expect, the standard deviation decreased as the size of the groups increased. However, we did not want group sizes that were too large, as it took a sizable amount of time to get each compass reading (roughly ten seconds per reading), and maps would take too long to construct if the group sizes were too large. We settled on group sizes of seven readings, since the averages of the groups of this size showed a standard deviation of 0.1 degrees, which was

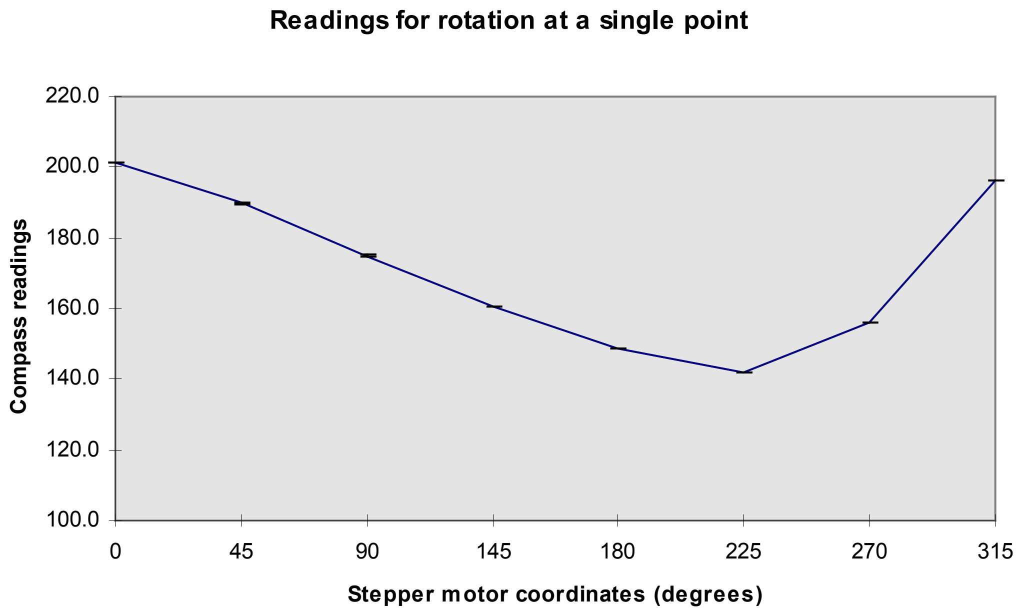

Figure 3: Results when the robot was rotated through 360 degrees at a single position. Rather than going through 360 degrees, the readings only swept through 60 degrees. Note the small size of the error bars.

small enough not to mask the spatial variations in the field. Also, this standard deviation of 0.1 degrees was found in the region of the lab that showed the most time variance, and fell to zero in areas with little time variance. So, all of the compass readings given in this lab are actually the averages of seven compass readings taken at single positions and orientations. This significantly reduced the effect of time-variance on our results.

The next experiment that we performed was to hook the compass up to our work station directly, rather than to the robot, and take readings by hand. This allowed us to get a rough feel for how the compass behaves, and the compass behaved exactly as the manual indicated. However, when the compass was actually mounted on the robot, it was discovered that turning the robot 180 degrees did not result in a compass reading change of 180 degrees, as it had when the readings were taken by hand. Rather, the changes in compass readings were significantly different from the actual changes in robot orientation. To get a quantitative idea of this, we had the robot take readings at multiple orientations at the same position. Our results are shown in figure 3. We found that the exact shape of this curve varied from position to position, which gave us one more means to distinguish between points when constructing our map. Because of this, we set up our mapping program to take readings at multiple orientations at each point.

We believe that this effect was due to the strong magnetic field produced by the robot's internal systems, which is somewhat stronger than the earth's magnetic field. As the robot changed its orientation, the direction or the field that its electronics produced changed as well. Fortunately, the robot's field was not so strong as to overwhelm the effects of the ambient field in the lab, and the spatial field variations were strong enough for our compass to detect them. The robot's field made it difficult to know which way was true north, but this was not relevant to our experiment, since the only information which was important to us was the field variation, not its absolute value.

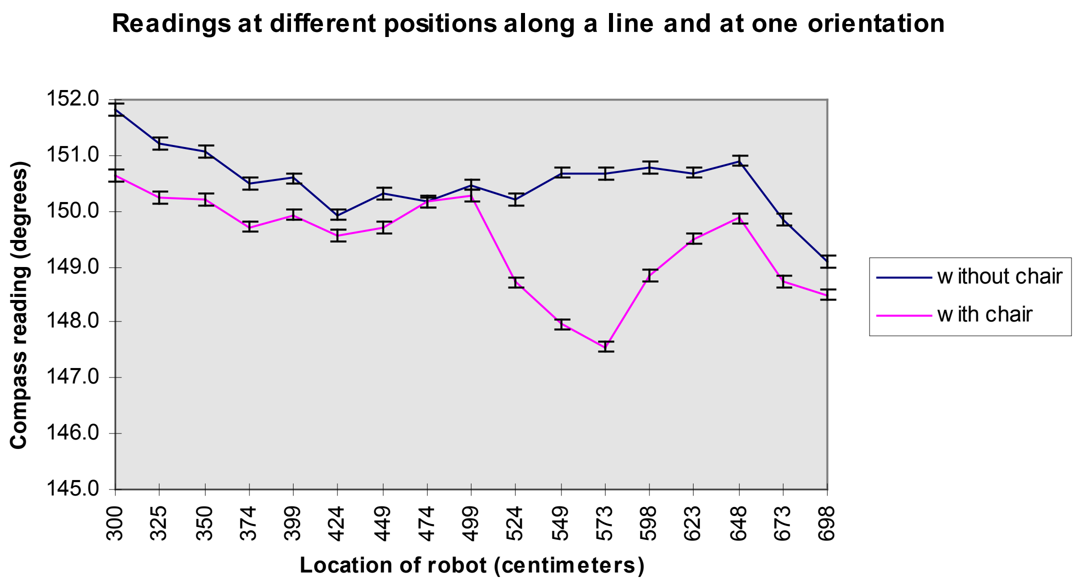

Figure 4: Compass readings when the robot was moved through the environment along a straight line and with no change in orientation. Two trials were done, and a metal chair was placed in the middle of the environment during the second trial. Note how this caused the readings to drop systematically almost everywhere, and how there was a large dip immediately near the chair.

Next, we moved the robot in a straight line and took data at a fixed orientation. This was done to give us an idea of the variation in the room's magnetic field at different positions. Our results are shown in figure 4. We were pleased to see that the deviation in the field between two consecutive points was over 0.1 degrees at all points but one (0.1 degrees being the maximum standard error measured earlier), and went well over 1.0 degrees several times. This was for point separations of 25 centimeters, which is relatively small, and the spatial variance of the field increased when the separation was increased. We then repeated the experiment, using the exact same path, but this time placing a metal chair near to the middle of the path. As can be see in figure 4, this had a very significant effect on our readings. This is both a good and bad thing. It means that simple magnetic objects could be used as beacons to introduce enough variation in environments with uniform magnetic fields (such as out of doors) so that our localization method would work. The bad side is that the robot is very sensitive to changes in its environment, so our method would not be suited in situations where a robot needs to operate in environments with many moving magnetically interesting objects, since the constantly changing magnetic field would render its map useless. We also found that the compass is very sensitive to electronics. For example, we discovered that turning a monitor off near the robot has a significant effect, as does speaking on a cellular phone.

2.5) Construction of maps

As mentioned earlier, the procedure by which the robot finds its orientation is to first construct a map of an environment and then, when it desires to know its location, take readings about the magnetic field and compare these readings to the map to find its location.

The mapping procedure which we used is as follows: the robot moves to equally spaced nodes in the environment, and takes compass readings at each node at different orientations. As discussed earlier, we found the best balance between quality of data and efficiency of data collection when measurements were used at four orientations per node and seven readings were taken and averaged out at each orientation.

This mapping procedure worked quite well for small maps, but it became apparent that

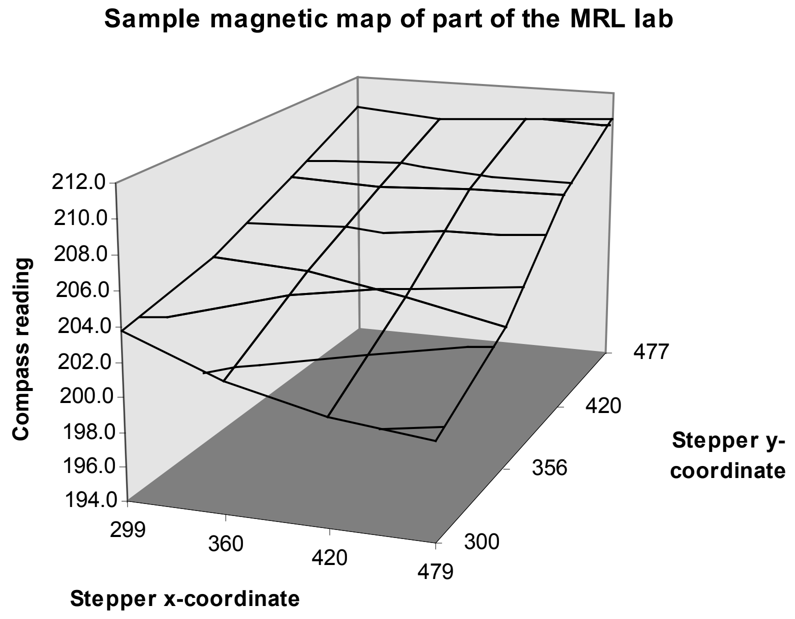

Figure 5: A magnetic field map of a 1.8 m x 1.8 m area of the MRL. Compass readings are only shown at one orientation, but the maps we used actually had four readings at each (x,y) point, each corresponding to a different robot orientation.

stepper motor error was compromising our results when larger maps were constructed. The purpose of the whole localization procedure is to have the robot take readings and compare them to its map so that it could correct for stepper motor, but this is obviously not possible when stepper error becomes significant in the map itself. One possible solution was to have the robot construct a small segment of the map, then return to the starting point where it could be manually repositioned, then have it go out to construct the next small segment of the map and so on. While this would certainly eliminate the stepper error, it is preferable to have a procedure where the robot could construct the map entirely without human interaction, so this method is a last resort.

We ended up noticing that whenever the robot rotated in one direction, the error systematically ended up being in that direction as well, and vice versa when it was rotated in the other direction. Since our original mapping program instructed the robot to move counter-clockwise whenever it rotated, it was no wonder that the stepper error accumulated. We modified our program to alternately rotate the robot clockwise and counter-clockwise, in the hope that the rotational stepper error would thus correct itself, and indeed, this turned out to be the case. Translational stepper error was eliminated in a similar fashion, by having the robot move alternately in different directions, since the direction of the translational stepper error was also systematic. It was found that, even for maps of the entire MRL testing area, the stepper error was essentially negligible when constructing maps. Of course, if the robot had been performing many movement in a non-systematic manner over an extended period of time, the stepper error would no longer be negligible, but the whole purpose of the map and localization procedure was to solve this problem.

We experimented with several grid sizes and densities. Obviously, as the grid size and the density of the nodes increase, so does the time that it takes to take data. Our localization procedure returns the coordinates of the closest node in the map, so a large node density increases the precision of the robots localization. Unfortunately, the stepper error is quite large for small movements, and the magnetic field variation is also small at small distances. This means that grid densities beyond are certain minimum are inaccurate enough to more than counteract the advantages of the increased precision. Very good results were achieved for node spacings of 30 centimeters and above, but our localization procedure sometimes mistook a node for one of its neighbors when smaller grid densities were used. For this reason and because it was considerably less time consuming, we constructed maps with node-spacings ranging from 30 centimeters to 100 centimeters.

We were limited to conducting experiments in the MRL lab, but we successfully mapped out areas ranging from one meter squared to the whole testing area, thus showing that our method could be used successfully in a wide variety of environment sizes. Figure 5 shows one of our maps plotted in space. Because it is very difficult to plot things in six dimensions, this figure only shows the compass readings taken at one particular orientation, but the maps we actually used contained data taken at four orientations at each node.

2.6) Localization procedure

The robot's position can be estimated once the map is constructed. After being placed at any point inside the space that has been mapped, the robot collects compass data at pre-specified orientations at the stationary point. The estimated position of the robot is that at which these magnetic readings are most consistent with those on the map, i.e. where the deviation in compass data between points of like orientation are minimum. The deviation is given by:

$$\text{dev} = [(c_1(y_1-x_1)^2 + c_2(y_2-x_2)^2 + c_3(y_3-x_3)^2 + \ldots + c_n(y_n-x_n)^2]^{1/2} \quad (1)$$

where:
i is the index representing robot orientation
x_i is the compass reading at the map node
y_i is the compass reading at the stationary point
c_i is a coefficient

Note that the robot has taken data at n orientations at each point.

The coefficients are used in order to take into account the different variances between compass data at different robot orientations. The coefficient that we used in the experiment is the standard deviation σ_i between elements of orientation i divided by the highest one $\max(\sigma_i)$. This process is repeated for each point on the grid. The closest point on the grid is estimated to be the point at which the deviation is smallest.

The following is an example of how this is done:

(x,y) Position (centimetres)	Robot Orientation (degrees)	Compass Reading (degrees)
(300,300)	360	205.5
	90	176.6
	180	148.0
	270	149.1
(400,300)	360	209.5
	270	142.5
	180	148.3
	90	178.8

TABLE 1A: grid data at two different nodes

Robot Orientation (degrees)	Compass Reading (degrees)
360	204.9
90	177.4
180	149.0
270	149.1

TABLE 1B: data at stationary point

Sample Calculation:

Position (300,300):

$$\text{Deviation} = [(204.9-205.5)^2 + (177.4-176.6)^2 + (149.0-148.0)^2 + (149.0-149.1)^2]^{1/2} = 1.38$$

Position(400,300) :

$$\text{Deviation} = [(204.9-209.5)^2 + (149.1-149.1)^2 + (149.0-148.3)^2 + (177.4-178.8)^2]^{1/2} = 4.86$$

Note: for simplification, all c_i are set to 1.

Since the deviation between the compass readings at the two points is less at position (300,300), we estimate the robot to be at (300,300) at the stationary point.

In order for our comparison of compass readings to work, the robot must start at the same orientation as in previous trials when taking its stationary magnetic data. In our case, the robot has been programmed to automatically initialize itself to the same orientation (relative to a stationary ring which does not turn relative to the ground) at the beginning of trial runs, so this did not turn out to be a problem.

In cases where the robot has been moved by some external means, the initialized orientation is no longer applicable. We can reinitialize the robot by simply moving the robot to an orientation that is characteristic of the initialized robot. In our case, they were around 200 degrees on the compass. Since we took readings at every 90 degrees, there was no overlap from compass data at other orientations to this particular compass reading. The robot would thus rotate until the compass read 200 degrees. While this method of initializing orientation is not too precise, it provides a good starting point for using the localization method for different environments.

2.7) Success rates

The following sample trial runs show how effective our localization procedure was at finding the location of the closest nodes in the map to the robot's actual position :

Actual Position (x,y) (centimetres)	Estimated Position (x,y) (centimetres)	Accuracy
(300,400)	(300,400)	good
(400,400)	(400,400)	good
(400,300)	(400,300)	good
(400,400)	(400,400)	good

Table 2: Success of localization procedure on a 100 cm x 100 cm grid with a 100 cm node separation, taken in an area of the MRL with a high spatial magnetic field variation. There was a 100% success rate in this experiment.

Actual Position (x,y) (centimetres)	Estimated Position (x,y) (centimetres)	Accuracy
(300,300)	(300,300)	good
(375,300)	(375,300)	good
(450,300)	(450,300)	good
(450,375)	(450,375)	good
(375,375)	(375,375)	good
(300,375)	(300,375)	good
(300,450)	(300,450)	good
(375,450)	(375,450)	good
(450,450)	(450,450)	good

Table 3: Success of localization procedure on a 150 cm x 150 cm grid with a 75 cm node separation, taken in an area of the MRL with a high spatial magnetic field variation. There was a 100% success rate in this experiment.

Actual Position (x,y) (centimetres)	Estimated Position (x,y) (centimetres)	Accuracy
(300,300)	(300,300)	good
(300,355)	(300,360)	good
(355,310)	(300,300)	no
(355,355)	(360,360)	good
(355,420)	(360,420)	good
(355,480)	(360,480)	good
(410,310)	(420,300)	good
(410,410)	(420,420)	good
(470,360)	(480,480)	no
(470,480)	(480,480)	good

Table 4: Success of localization procedure on a 180 cm x 180 cm grid with a 60 cm node separation, taken in an area of the MRL with a high spatial magnetic field variation. There was a 80% success rate in this experiment.

Actual Position (x,y) (centimetres)	Estimated Position (x,y) (centimetres)	Accuracy
(300,300)	(300,300)	good
(300,340)	(300,330)	good
(337,340)	(360,300)	no
(330,360)	(330,360)	good
(360,330)	(360,330)	good
(360,360)	(360,360)	good

Table 5: Success of localization procedure on a 60 cm x 60 cm grid with a 30 cm node separation, taken in an area of the MRL with a high spatial magnetic field variation. There was a 83% success rate in this experiment.

Actual Position (x,y) (centimetres)	Estimated Position (x,y) (centimetres)	Accuracy
(300,300)	(450,300)	no
(300,375)	(300,375)	good
(300,450)	(300,450)	good
(375,300)	(450,300)	no
(375,375)	(450,375)	no
(375,450)	(375,450)	good
(375,375)	(375,375)	good
(450,300)	(450,300)	good
(450,375)	(450,375)	good
(450,450)	(375,450)	no

Table 6: Success of localization procedure on a 150 cm x 150 cm grid with a 75 cm node separation, taken in an area of the MRL with a low spatial magnetic field variation. There was a 60% success rate in this experiment.

Actual Position (x,y) (centimetres)	Estimated Position (x,y) (centimetres)	Accuracy
(300,320)	(300,300)	good
(300,400)	(300,400)	good
(300,600)	(300,600)	good
(400,400)	(400,400)	good
(400,500)	(500,400)	no
(500,435)	(300,400)	no
(580,320)	(600,300)	good
(600,400)	(600,400)	good
(600,600)	(600,600)	good
(640,410)	(600,400)	good
(680,320)	(700,500)	no
(700,500)	(700,500)	good

Table 7: Success of localization procedure on a 400 cm x 200 cm grid with a 100 cm node separation, taken throughout the whole MRL testing environment. There was a 75% success rate in this experiment.

Observations:

- Our localization method is more effective at larger node separation distances, but still works accurately for the most part when separations as small as 30 cm are used.
- Our method works significantly better at places where there is a high amount of spatial variation in the compass data
- The method estimates the position to within 1 node of the correct position almost all of the time, even when an incorrect position is returned.

We can conclude that this method should be used in places where there are a large amount of stationary magnetically interesting objects, and not in a large empty space. This method also very reliably places the robot to a point in the room with a precision of about 75cm, provided there is enough spatial variation .

2.8) Possible Improvements

One method that should make our results even more accurate in theory would be to move the robot to neighboring points in program "STATIONARY". After estimating its position in the manner used up until now by spinning in place, it would move to neighboring nodes and check to see if their positions correspond to the neighboring nodes of the estimated position on the map. However, due to time constraints, we have not been able to explore the effectiveness of this method.

The compass data can also be made more precise if more readings are taken at each node. However, this is not very feasible since the batteries of the robot do not last very long, and it is not possible to make maps that take excessive amounts of time to make.

3) Sonar method

As mentioned earlier, there was not enough time to implement this method, but it is still worth mentioning as an area for future groups to research and because it would be useful in environments where the compass method would not work, such as where there is little magnetic variation or places where there are many mobile magnetically interesting objects.

Sonar works by emitting pulses of sound and measuring the time that it takes for the pulse to return to the sensor after being reflected off of an object. Since the speed of sound in air is known, it is thus easy to calculate the distance to the object that is reflecting the pulses. The Nomad 200 is equipped with a sonar that allows it to take sonar readings in 360 degrees simultaneously, thus allowing it to acquire information about all of its surroundings at once. Ideally, this would allow the robot to know how far objects are from it at all times. Unfortunately, there are serious problems when sonar is used in enclosed

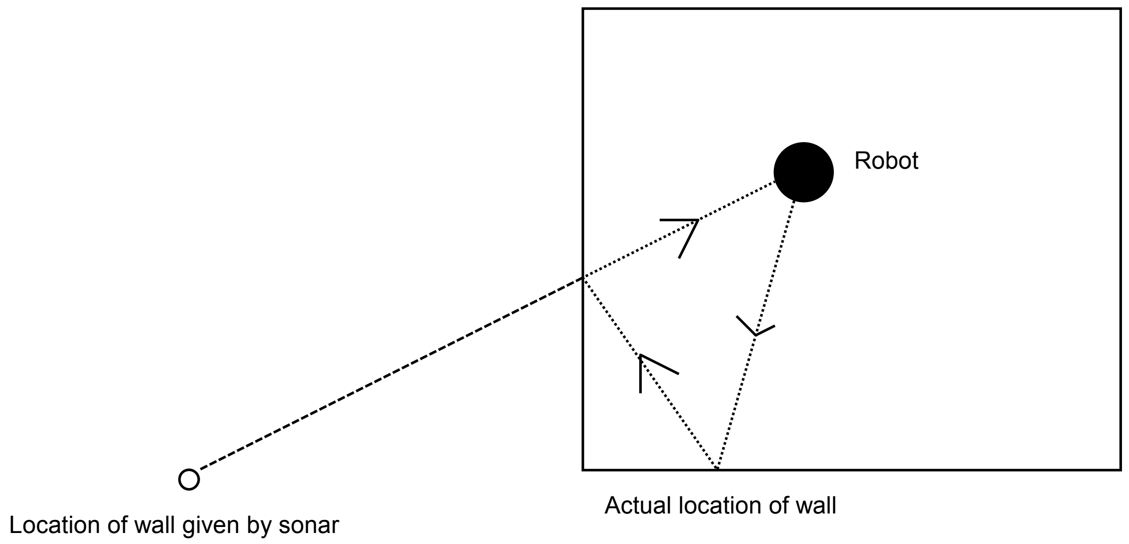

Figure 6: An example of how sonar can give faulty readings when used in enclosed areas.

spaces. First of all, nearby objects can cast a long sonar shadow, thus obscuring large parts of the environment from the robot's sensors. More seriously, a pulse can be reflected off walls multiple times before returning to a sensor, thus giving the sonar faulty readings as to the position and distance of the wall. An example of this is shown in figure 6.

The way to get around this problem is to instruct the robot to ignore all readings that do not return after a short period of time. This would stop all pulses that return after multiple reflections from being counted. The trade-off is that the robot is then incapable of knowing about objects that fall outside its region of acceptance. Fortunately, this is not important for the localization method that we propose.

Just as in the compass method, we begin by making a map of the environment. We have the robot simply move forward until it comes encounters a wall, at a distance of perhaps one meter or so, which is more than close enough to fall within its region of acceptance. It then rotates so that it can move parallel to the wall and follows the wall until it finds a corner, at which point it positions itself so that it can follow this new wall. It proceeds in this manner until it has returned to the place where it first started following the walls. All this while, the robot keeps a record of the lengths and relative orientations of each wall. Once this has been done, the robot has completed its map. Assuming that the environment is not immense, this procedure would involve sufficiently few movements for stepper errors to become significant in the mapping.

At later points, whenever the robot wants to know its location, it could move to a wall and follow it until it is able to match the length and orientations of the walls uniquely to part of its map. It will then know where it is in the room, and can then move to any other desired point in the room using its stepper motors. This method requires a certain amount of asymmetry to work, of course. If the robot were operating in a square room, for example, it would have no way of distinguishing one wall from another. The solution to this is to use the sonar method in conjunction with some other measuring device. For example, assuming that the magnetic field of the robot is not so strong as to overpower the earth's field, a compass could be used to determine which general direction is north. As long as the direction of the earth's field is shifted less than 45 degrees, the robot will be able to distinguish the north wall of a square room from any of the other walls. It can then follow this wall until a corner is reached, at which point it will know exactly where it is. This method will not work in areas with no corners, such as circular rooms, or outdoors where there are no walls, but it should be quite effective in other environments.

It should be noted that the robot will need to be able to distinguish between walls and small objects such as other robots for this method to work. This can be done by having the robot only treat objects of a certain minimum size as walls. If they do not reach this minimum, then the robot will simply move around them and proceed until it encounters something which meets this minimum size, whereupon it will start constructing its map or begin its localization procedure. This also prevents the map from incorporating potentially mobile objects into its map.

4) Conclusions

This experiment was intended as a study of the feasibility of the compass method in robot localization, and as such it was a success. We showed that there is indeed enough variation in the magnetic field of the room that we studied for the robot to determine its location with very good reliability.

Our success shows that there is reason to investigate this area further, since there is a great deal of room for future experimenters to build on our work. For example, they could use different robots to see how well our method works with them. A robot that was built with the intent of using these types of measurements could surely be designed to generate a much weaker magnetic field, and thus allow the compass to take more precise measurements of the room's magnetic field.

It would also be interesting to see if there would still be enough variation in the magnetic field in other types of environments. One could see if our method works out of doors, or could specifically design an environment using magnetic beacons so that the robot could be even better able to find its position. And, of course, the effectiveness of the sonar method could certainly be explored.

There are limitations to the usefulness of the compass method, just as there are to any localization method. It probably would not work outdoors or in the middle of a large room, since there is very little spatial field variation in these environments. It does not work in situations where there are mobile magnetically interesting objects or intermittently operating electronics which cause the magnetic field to change non-periodically. However, we showed that the compass method can be quite successful in environments where this is not the case.

Another limitation of the compass method is that it is limited in precision by the density of the map nodes, since the localization procedure returns the location of the nearest node. All of our positional determinations thus have an error of one half of the node separation. This is a problem where very precise locations are needed. However, we obtained excellent results with node separations of 60 centimeters, and could have obtained even better resolution if we had had better stepper motors, since the ones that we used have relatively large error when moving at distances much smaller than this. Our method will work perfectly fine in situations where a great deal of precision is not needed, especially since the positional error of one half the node distance does not accumulate each time a positional determination, since the old coordinates are thrown out when the new determination is made.

We were very pleased to have success with this method, as previous researchers had considerably more trouble in localizing the robot. Our results show future researchers with more resources and time than we had could certainly develop our work here into something with considerable practical value, and we hope that future groups will build on our results.

5) Bibliography

Cox, I. Wilfong G., *Autonomous Robot Vehicles*, New York: Springer - Verlag, 1990.

Kjellstrom, B., *Be an Expert With Map and Compass: The Orienteering Handbook*, Harrisburg (Pa): Stackpole Books, 1967.

Lee, D., *Map-Building and Exploration Strategies of a Simple Sonar Equipped Robot: An Experimental Evaluation*, Cambridge: Cambridge University Press, 1996.

Press, W., *Numerical Recipes in C,* Cambridge: Cambridge University Press, 1992.

Appendix 1: Code Listings

MAP:

```
/* MAP.C: builds map and obtains robot's compass data  */

#include <stdio.h>
#include <time.h>
#include <math.h>
#include "RD10.h"

FILE *outfile;

/*************************************/
void delay()
     /* delays readings by specified time amount
        1600 iterations corresponds to 1s
        650 000 clicks corresponds to 1s*/
{  int tinitial;
   int x;
   x= 1;
   tinitial = clock();
   while (x < 3900000)
     {  x = clock();
     }
}
/*************************************/
int getreading(int nread)
/* gets compass readings */
{  char reading;

   int i, j;

   FILE *inputfile, *magfile;

   magfile = fopen("magnet.dat", "w");
   for (j=1;j<=nread;++j)
   {  inputfile = popen("c2", "r");
      for (i=1;i<=25;++i)
      { fscanf(inputfile,"%c",&reading);
        if (i>7 && i<13) fprintf(magfile,"%c",reading);
      }
      pclose(inputfile);
      fprintf(magfile," ");
   }
   fclose(magfile);
   return(0);
}  /* getreading */

/*************************************/
float compass(int nread)
/* puts compass data into data file */
{ FILE *magfile;
  float data;
  float sum;
  float mean;
  int j;
  getreading(nread);
  magfile = fopen("magnet.dat","r");
  sum = 0;
  for (j=1;j<=nread;++j)
  { fscanf(magfile,"%f",&data);
    printf("%f\n",data);
    sum = sum+data;
    mean = sum/nread;
  }

  fclose(magfile);
  return(mean);
} /* compass */
```

```
/*****************************************/
int main(void)
{ RD_Pose_Ptr place;

  char filename[10];
  char *hostname = "voyager.cim.mcgill.ca";
  char *robot = "invader";
  char *socket = "5090";

  RD_Robot_Type Robot_Type = RD_NOMAD;
  float mag;
  float *xmove;
  float *ymove;
  float *rotate;
  float node_distance, x_distance, y_distance, radial_readings,  degree_interval;
  float x_travelled=0, y_travelled=0,angle_rotated = 0;
  float angle;
  int cwise;
  int orient;
  float reading;
  int nread;

  xmove = (float *) 180;
  ymove = (float *) 180;
  rotate = (float *)180;

  printf("Hostname?\n");
  printf("Filename:");
  scanf("%s", filename);

  outfile = fopen(filename, "w");

  printf("Connecting to Robodaemon...\n");
  RD_Connect(hostname, socket);
  printf("Connection complete.\n");

  printf("Creating a robot...\n");
  RD_Create(Robot_Type, (char *) robot, (char *) NULL);
  printf("Creation complete.\n");

  place = (RD_Pose_Ptr)malloc(sizeof(RD_Pose));
  RD_Get_Pose((char *) robot, (RD_Pose_Ptr) place);
  printf("Pose1 is (x,y,theta)=(%f,%f,%f)\n", place->X, place->Y, place->Theta);

  printf("Distance between nodes (m): ");
  scanf("%f", &node_distance);
  node_distance = node_distance * 100;
  printf("X-distance: ");
  scanf("%f", &x_distance);
  x_distance = x_distance * 100;
  printf("Y-distance: ");
  scanf("%f", &y_distance);
  y_distance = y_distance * 100;
  printf("Radial readings: ");
  scanf("%f", &radial_readings);
  degree_interval = 360 / radial_readings;
  printf("Number of compass readings per point?");
  scanf("%i", &nread);

  orient = 1;
  printf("Moving the robot...\n");
  cwise = 1;
  angle_rotated = 0;
    while (angle_rotated < 360)
     {  delay();
        printf("Hoiles!");
        RD_Get_Pose((char *) robot, (RD_Pose_Ptr) place);
        if (place->Theta < 20 ) place->Theta = place->Theta+360;
        fprintf(outfile,"%f %f %f ", place->X, place->Y, place->Theta);
        reading = compass(nread);
        fprintf(outfile,"%f\n",reading);
        RD_Rel_Rotate((char *) robot, degree_interval, rotate);
        angle_rotated = angle_rotated + degree_interval;
     }
  while (y_travelled <= y_distance)
   { x_travelled = 0;
     while (x_travelled < x_distance)
      {
        RD_Translate((char *) robot, node_distance, (char *) NULL,rotate);
        angle_rotated = 0;
        cwise = (-1)*cwise;
        while (angle_rotated < 360)
         {  delay();
            printf("Hoiles1!");
            RD_Get_Pose((char *) robot, (RD_Pose_Ptr) place);
            if (place->Theta < 20) place->Theta = place->Theta+360;
            fprintf(outfile,"%f %f %f ", place->X, place->Y, place->Theta);
            reading = compass(nread);
            fprintf(outfile,"%f\n",reading);
            RD_Rel_Rotate((char *) robot, cwise*degree_interval, rotate);
            angle_rotated = angle_rotated + degree_interval;
          }/* angle rotated */
        x_travelled = x_travelled + node_distance;
      } /* x-travelled */
     if (orient == 1) angle = 90;
     else angle = (-1)*90;

     if (y_travelled < y_distance)
      {  RD_Rel_Rotate((char *) robot, angle, rotate);
         RD_Translate((char *) robot, node_distance, (char *) NULL,rotate);
         RD_Rel_Rotate((char *) robot, angle, rotate);
         cwise = (-1)*cwise;
         angle_rotated = 0;
         while (angle_rotated < 360)
         {  delay();
            printf("Hoiles2!");
            RD_Get_Pose((char *) robot, (RD_Pose_Ptr) place);
            if (place->Theta < 20) place->Theta =  place->Theta + 360;
            fprintf(outfile,"%f %f %f ", place->X, place->Y, place->Theta);
            reading = compass(nread);
            fprintf(outfile,"%f\n",reading);
            RD_Rel_Rotate((char *) robot, cwise*degree_interval, rotate);
            angle_rotated = angle_rotated + degree_interval;
         } /* while */
      }  /* if */
     y_travelled = y_travelled + node_distance;
     if (node_distance == 0) y_travelled = y_distance + 1;
     orient = (-1)*orient;

   } /* while */

  printf("Done\n");

  RD_Get_Pose((char *) robot, (RD_Pose_Ptr) place);
  printf("Pose2 is (x,y,theta)=(%f,%f,%f)\n", place->X, place->Y,
       place->Theta);
  RD_Destroy(robot);
  RD_Disconnect();
  fprintf(outfile,"-1000\n");

  fclose(outfile);
  printf("Program complete...\n");

  return(0);
} /* main */
```

Analzyze:

```
/* program ANALYZE.C : analyzes stationary data and grid data,
    estimates robot's present position */

#define MAXREADINGS 15
#define MOTOR_ERROR 10
#include <stdio.h>
#include <time.h>
#include <math.h>

FILE *compfile,*gridfile;

struct grid {
  float x; /* x coordinate */
  float y; /* y coordinate */
  float o1; /* robot orientation */
  float c1; /* compass orientation */
};
struct point {
  float o2;
  float c2;
};

struct coord {
  float x;
  float y;
};
/*************************************/
int round(float x)
/* round number to nearest integer */
{  int num;
   num = (x+0.5)/1;
   return(num);
}

/*************************************/
double power(float x, int n)
{ int i;
  double pow;

  pow =1;
  for (i=1;i<=n;++i)
   { pow = pow*x;
   }
  return(pow);
}

/*************************************/
int stdev (int nread,  char gfilename[12], double dev[20])
/* gets standard deviation of each orientation */
{ double sumsquares[50], sum[50];

  FILE *gridfile;
  struct grid gdata;

  double mean;
  double num[50];
  int i;
  int ind;
  gridfile = fopen(gfilename,"r");

  gdata.x = 0;

  for (i=0;i<nread;++i)
  { sumsquares[i] = 0;
    sum[i] = 0;
    num[i] = 0;
  }

  while(gdata.x > -500)
  { fscanf(gridfile,"%f%f%f%f", &gdata.x, &gdata.y, &gdata.o1, &gdata.c1);

   if (gdata.x > -500)
   {  if (sqrt(power(gdata.o1-360,2))<MOTOR_ERROR)
      { sumsquares[0] += gdata.c1*gdata.c1;
        sum[0] += gdata.c1;
        num[0] = num[0]+1;
      } /* if */
      else
      { ind = round(gdata.o1/(360/(float) nread) );
        if (sqrt(power(gdata.o1-ind*(360/(float) nread),2)));
        { sumsquares[ind] += gdata.c1*gdata.c1;
          sum[ind] += gdata.c1;
          num[ind] = num[ind]+1;
        }
      }   /* else */
   } /* if */
  } /* while */
  if (num[0] == 0) printf("Error:no data");
  else
   { for (i=0;i<nread;++i)
     { mean = sum[i]/num[i];
       dev[i] = sqrt((sumsquares[i]-num[i]*mean*mean)/num[i]);
     }
   }

  fclose(gridfile);

  return(0);
} /* stdev */

/*************************************/
int coefficient(int nread,double stdev[20],double coeff[20])
/* gets coefficient for each orientation */
{ int i;
  double highest_coeff;

  highest_coeff = 0;
  for (i=0;i<nread;++i)
   { coeff[i] = power(stdev[i],1);
     if (coeff[i] > highest_coeff) highest_coeff = coeff[i];
   }

  for (i=0;i<nread;++i)
  { if (highest_coeff != 0) coeff[i] = coeff[i]/highest_coeff;
  }

  return(0);
} /* coefficient */

/*************************************/
int main (void)
{ char filename[20],gridfilename[20];
  int i,j, nread, pt;
  double diff, mindiff;

  double dev[50],coeff[50];

  struct grid griddata;
  struct point pointdata[MAXREADINGS];
  struct coord closestpt;

  printf("Compass data filename (at present robot position): ");
  scanf("%s", filename);
  printf("Compass grid data filename: ");
  scanf("%s", gridfilename);

  printf("Number of compass readings per point? ");
  printf("%s %s\n",filename,gridfilename);
  scanf("%i", &nread);
  mindiff = 500;

  gridfile = fopen(gridfilename,"r");
  compfile = fopen(filename,"r");

  stdev(nread, gridfilename, dev);

  coefficient(nread,dev,coeff);

  for (j=0;j<nread;++j)
  { fscanf(compfile,"%f%f",&pointdata[j].o2,&pointdata[j].c2);
  }

  griddata.x = 0;
  while (griddata.x > -500 )
   { diff = 0;
      for (i=0;i<nread;++i)
       {  fscanf(gridfile,"%f%f%f%f", &griddata.x, &griddata.y,
            &griddata.o1, &griddata.c1);
          pt= 0; /* get orientation on point corresponding to 1st orientatio
            on grid */
          while (sqrt(power(griddata.o1-pointdata[pt].o2,2))> MOTOR_ERROR  && pt
                <= nread)
            ++pt;
          diff = diff + coeff[pt]*power(griddata.c1-pointdata[pt].c2,2);
       } /* for */

     if (diff < mindiff && griddata.x >-500)
       { mindiff = diff;
         closestpt.x = griddata.x;
         closestpt.y = griddata.y;
       } /* if */
   } /* while EOF */

  printf("closest pt:%f %f\n",closestpt.x,closestpt.y);

  fclose(gridfile);
  fclose(compfile);

  return(0);
} /* main */
```