IMPLEMENTING A ROBUST 3-DIMENSIONAL EGOCENTRIC NAVIGATION SYSTEM

By

Paul Fleming

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

August, 2005

Nashville, Tennessee

Approved:

Dr. Kazuhiko Kawamura

Dr. Mitch Wilkes

Dedicated to:

Kevin Fleming, who always encouraged and supported me,

Jimmy Fleming, who shared so much happiness,

and to Anne Fleming, who is always listening and supportive

ACKNOWLEDGEMENTS

Thanks to Dr. Kawamura for his guidance,
to Dr. Wilkes for many hours of help, instruction, feedback,
to Katherine Achim for much help and support
and to everyone at the CIS lab, for all the help

TABLE OF CONTENTS

Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Robot Navigation is a large component of current robotics research. Much time and effort has been expended in determining effective methods for getting robots from one location to another. In the book *Artificial Intelligence and Mobile Robotics*, [Kortenkamp, 1998], explains that the three central questions of robot navigation are:

Where am I?

How do I get to other places from here?

Where are other places relative to me?

In answering these questions, many algorithms have been designed which vary in technique. Earlier designs tended to attempt to metrically map out the robot's environment and then record the robot's movement through odometry. [Kortenkamp, 1998] Later, "sense-act" algorithms were introduced, in which the robot behaves in an "insect-like" manner; responding directly to sensor data with little or no internal deliberation (map-making, planning etc.) [Kortenkamp, 1998] Still later algorithms tried to blend the two approaches into a Hybrid, which established low-level behaviors under the control of a more deliberative component. [Murphy, 2000]

One type of robot navigation is a perception-based "sense-plan-act" approach, in which a robot perceives, using cameras, sonar etc., a goal and obstacles and then attempts to move toward the goal while avoiding the obstacles. In this type of robot navigation, which could be called "perception-based", the process of navigating can often be broken down into

two competing tasks. The first is getting the robot to some goal location. The other task is getting the robot to avoid obstacles. [Murphy 2003]

In this paper, a method to implement the "go-to-goal" aspect of navigation is discussed. Specifically, a method called Egocentric Navigation is reviewed. Then improvements toward making the system more accurate and efficient will be proposed. Finally, the experimental and simulation results of this new Egocentric Navigational system follow.

In total, this paper will lay out a complete system for implementation on a robot which will accomplish navigation to goal in a landmark-based "egocentric" manner. Sensing and perceiving the environment egocentrically involves viewing the environment not in a world scope; as in "that object is at a certain latitude and longitude, or location in the room." Instead, the world is perceived and analyzed only in reference to the robot itself; as in, "that object is in front of me" [Murphy 2003].

The advantages of egocentric over a world, or "allocentric" model are as follows. First, it is more natural for the robot to perceive egocentrically, because data returned from on-board sensors is intrinsically egocentric (save GPS data). Additionally, it could make robots more human-like in their approach to navigation. Instead of instructing a robot to go "100 meters and turn", you may say, "go to the sign and turn", which the robot could hear as "move until my sensors perceive the sign is at my side and then turn", which is a much more natural way of giving directions. [Kawamura, 2002] [Koku, 2003]

CHAPTER II

BACKGROUND

Overview

This section of the paper reviews Egocentric Navigation as outlined in [Koku, 2003]. In order to do this, we must start with a discussion of the Sensory Egosphere; witch is the means by which sensory data is stored egocentrically, in the current and the proposed system. Then we will proceed into discussing egocentric navigation, and will finish by analyzing and finding areas of egocentric navigation that could be improved.

The SES

In order to perform navigation, the robot needs an internal memory structure to represent knowledge of where it believes itself to be, and where it "wants" to be. It can then perform calculations using this stored information to determine which direction it should go to arrive at a goal.

For the case of egocentric navigation the Sensory Egosphere and the Landmark Egosphere are the memory structures employed. These structures are based on the egosphere, which was originally conceived by J.S. Albus [Albus, 1991]. He defined an Egosphere as such:

> An Egosphere is a 2-Dimensional (2D) spherical surface that is a map of the world as seen by and observer at the center of the sphere. Visible point on regions or objects

in the world are projects on the Egosphere wherever the line of sight from a sensor at the center of the egosphere to the points in the world intersect the surface of the sphere. Egosphere coordinates are thus polar coordinates defined by the self at the origin. As the self moves, the projection of the world flows across the surface of the egosphere. [Peters, 2001]

Below is a visualization of the egosphere.

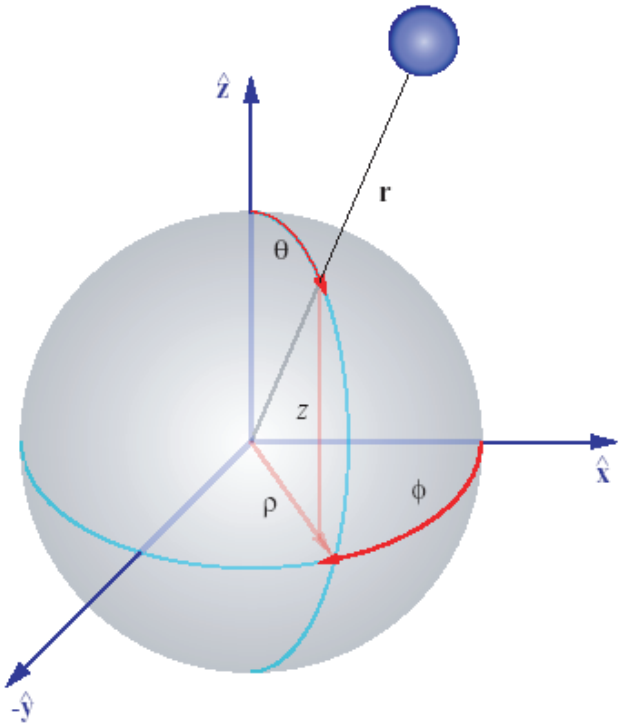

Figure 1. Visualization of the Egosphere [Peters, 2001]

The blue ball represents an object in the environment, while the origin of the axis represents the robot, or more specifically the robot's sensor location. The egosphere stores the location of the ball as occurring at the angular location on the sphere where the line of sight intersects the surface of the sphere. In the figure above, this location is phi, theta. However, it is clearer to refer to them as the azimuth and elevation angles [Wilkes].

The Sensory Egosphere is a discrete version of the egosphere discussed above. Because the sensors of a robot are by nature discreet, the Sensory Egosphere uses not a true sphere, but a geodesic dome. Each of the nodes on the dome represents a storage location in a database. An object is stored in the Sensory Egosphere at the closest node to its true azimuth and elevation angles [Peters, 2001].

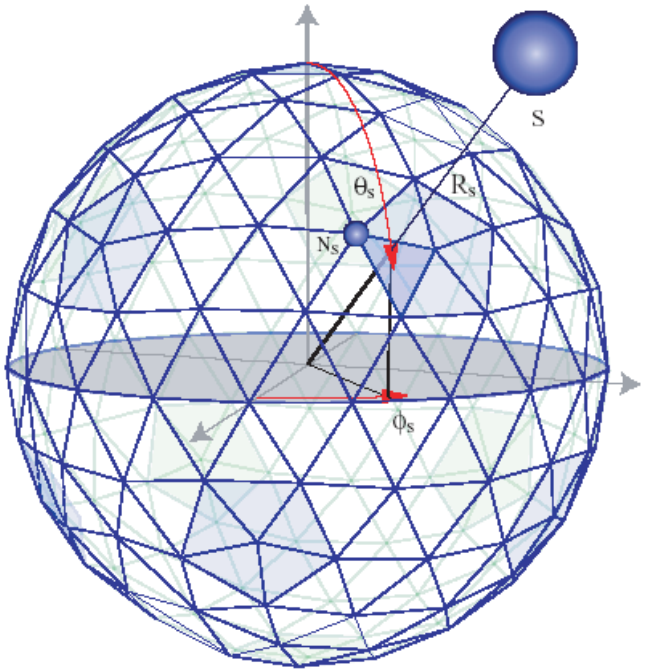

Figure 2. Visualization of a Sensory Egosphere [Peters, 2001]

Here we see the same object in the world, but now it will be mapped to node N_s, as this is the closest node to its true location [Peters, 2001].

The Sensory Egosphere (SES) can then be used in a variety of capacities, such as sensor fusion. It represents a very natural way to store sensory information. [Peters, 2001]

In this thesis, the SES only holds visual information. Specifically, only the locations of found landmarks will be stored in the SES for my experiments.

The "Landmark Egosphere" (LES) is very similar to the SES. The LES is essentially a special case of the SES. It contains the positions of an egosphere of landmarks as viewed from some location. This leads to the SES being thought of as the perceived current environment (stimuli), or short-term memory [Hunstberger, 2001], and the LES being thought of as stored, or long-term memory. In other words, the SES represents the environment as it appears now while the LES represents the environment at other locations [Kawamura, 2002].

In summary, egospheres represent a way to store (for short and long-term) the robot's knowledge of its surroundings. They represent not a metric map, but an egocentric mapping, of the robot's environment. Returning then to the 3 questions posed at the introduction, we've answered two of the questions. In response to the first question (Where am I?): I (the robot) am at the place that corresponds to this current SES. Other places (Where are other places relative to me?) are at locations given by their respective LES. How to get to other places from here (the second question) can now be answered through "Egocentric Navigation".

Egocentric Navigation

If we then build upon the concepts of the SES and the LES, we could imagine a scenario in which the LES is the perceived egosphere at some goal region, while the SES is the current state of the robot. An idea for navigation can be to move the robot in such a way as to reduce the difference between the SES and the LES that is the current and desired egosphere. In this way, the robot moves in a go-to-goal fashion.

In Bugra Koku's doctoral dissertation, one method to accomplish this is outlined [Koku, 2003]. This method is called Egocentric Navigation; this is because the robot is not moving based on a map, or GPS or some other "global" system, but on its own personal, or "egocentric", perception of the world. The dissertation lays out a navigational system which, given an SES (current state) and an LES (goal state) provide a direction of travel which will move the robot toward the goal and thus reduce the difference between the SES and the LES.

One important note is that for this method of navigation, only artificially made landmarks are recognized as objects and placed on the SES or the LES. This is done to simplify visual recognition. Therefore the SES and the LES contain information only about these landmarks, which is why Egocentric Navigation is described as "landmark based".

The first step in Egocentric Navigation (as computed in [Koku, 2003]) is to simplify the SES. Projecting the "egosphere information onto the equatorial plane of the sphere, resulting in a 2D representation, does this. This simplified SES and LES representations are all that are currently needed for navigation." [Kawamura, 2002]

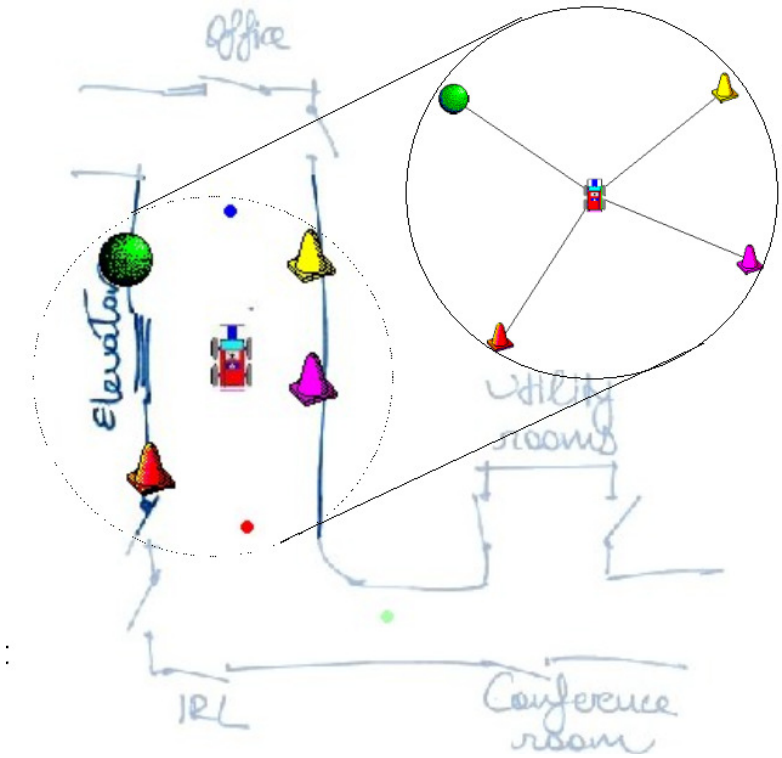

Figure 3. Perceiving 2D SES [Kawamura, 2001]

Above we see a simulated robot perceiving the landmarks in its environment and then storing that memory into a 2D SES. These are simplified egospheres, in that now the locations of the landmarks are described by the azimuth angle alone. However, it is still useful to think of these as 2D egospheres in the sense that the landmarks are all projected to lie on a 2-Dimensional plane, rather than 3-Dimensional space. For this reason these will be called 2D egospheres, and the original egospheres 3D.

Now, the Egocentric Navigation algorithm will give directions for the robot to move which will reduce the difference between the SES and the LES. The method employed is to first filter out all the landmarks that are not common to both the SES and the LES. Next compare the angles between every pair of landmarks on the SES with the angles of those

same pairs on the LES. For each pair of landmarks, assign a unit vector along the bisector of the angle between them and have it point either towards or away from the landmarks depending on the result of the angular comparison [Kawamura, 2002]. So for instance consider the egospheres below.

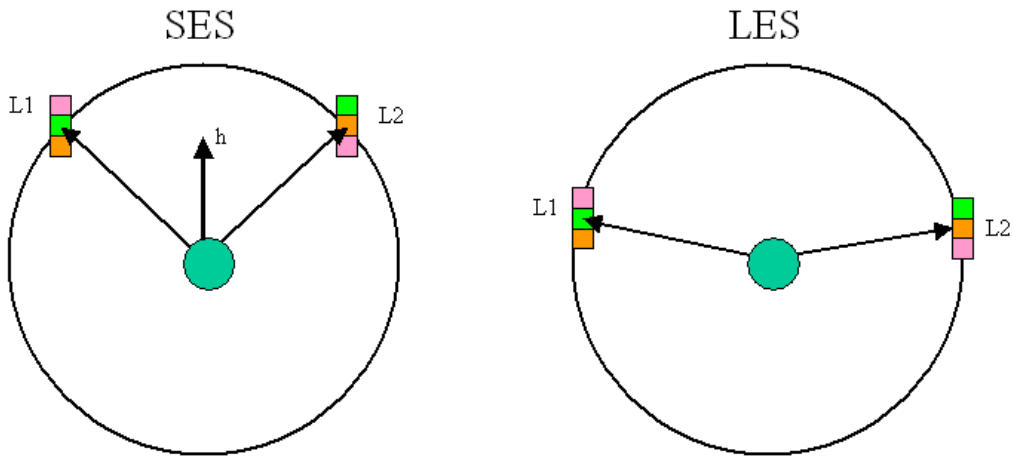

Figure 4. 2 Dimensional SES and LES

In this example, the angle between the two landmarks is larger in the LES, so intuitively the heading for the robot (**h**) is selected to move toward the landmarks along the bisector. Had there been a third landmark on either egosphere, which did not appear on the other, it would not be used in the calculation. If however there were a third landmark, which appeared on both egospheres, then it would also be used in the calculations. In this case, there would be three pairs of angles to work with, and therefore three resulting headings. Egocentric Navigation calculates all three headings, and then computes the vector sum of

these headings to derive a final heading. This logic continues into the cases of four shared landmarks and up.

Egocentric Navigation can be formally defined now for any number of landmarks. The definition will be described using vector mathematics. First define the unit vector $\underline{\mathbf{u}}^c_i$ as the unit vector pointing to a landmark i on the SES, while $\underline{\mathbf{u}}^t_i$ points to a landmark on the LES. Now we can follow through with the following calculations

$$d^c_{ij} = \underline{\mathbf{u}}^c_i \cdot \underline{\mathbf{u}}^c_j \qquad \underline{\mathbf{C}}_{ij} = \underline{\mathbf{u}}^c_i \times \underline{\mathbf{u}}^c_j$$
$$d^t_{ij} = \underline{\mathbf{u}}^t_i \cdot \underline{\mathbf{u}}^t_j \qquad \underline{\mathbf{T}}_{ij} = \underline{\mathbf{u}}^t_i \times \underline{\mathbf{u}}^t_j$$

$$A_{ij} = \mathrm{sgn}(d^c_{ij} - d^t_{ij})$$
$$B_{ij} = [\mathrm{sgn}(\underline{\mathbf{C}}_{ij} \cdot \underline{\mathbf{T}}_{ij}) + 1] / 2$$

$$\underline{\mathbf{u}}_{ij} = (1 + B_{ij}(A_{ij} - 1))(\underline{\mathbf{u}}^c_i + \underline{\mathbf{u}}^c_j / \| \underline{\mathbf{u}}^c_i + \underline{\mathbf{u}}^c_j \|)$$
$$\underline{\mathbf{h}} = \Sigma \underline{\mathbf{u}}_{ij} \quad \text{where } i \neq j$$

[Koku, 2002]

In the above calculations, "A_{ij} represents the relative magnitudes of the angles between the pairs from the SES and LES". While "B_{ij} represents the ordering of the pairs". $\underline{\mathbf{u}}_{ij}$ is the resultant heading for each pair, and finally $\underline{\mathbf{h}}$ is the final heading. [Kawamura, 2001].

The above calculations can be used to determine direction for any number of landmarks greater then one. The case of a single landmark is a special case where the heading is set to be towards the landmark. This is done in the hopes that by moving towards the only visible landmark, more landmarks will be discovered [Koku, 2003].

This completes the description of Egocentric Navigation as outlined in [Koku, 2003]. In this paper, a new robot navigational system which employs full 3-Dimensional egospheres will be designed. For clarity then, the above method will henceforth be referred to as 2-Dimensional Egocentric Navigation (2D ENav).

Two Dimensional Egocentric Navigation Analysis

The dissertation "Egocentric Navigation and its Applications" by A. Bugra Koku, provides a detailed analysis of 2D ENav. Specifically, the paper examined case by case, how does the algorithm perform given varying start and goal regions, quantity and locations of landmarks etc. These were tested in each scenario would a robot running 2D ENav reach the goal (tested convergence), and would it do so as quickly as possible, i.e. a straight line (optimality of path). To do this, a simulation program called ESim was used extensively. ESim allows simulation and visualization of 2D ENav by a number of methods.

One simulation method ESim provides is a visualization showing the heading calculated for a number of points on a simulated map. For instance, a user could place several landmarks in a scenario along with a goal region, and ESim would demonstrate what the headings would be in different positions around the area based on ENav calculations done at the point in the scenario.

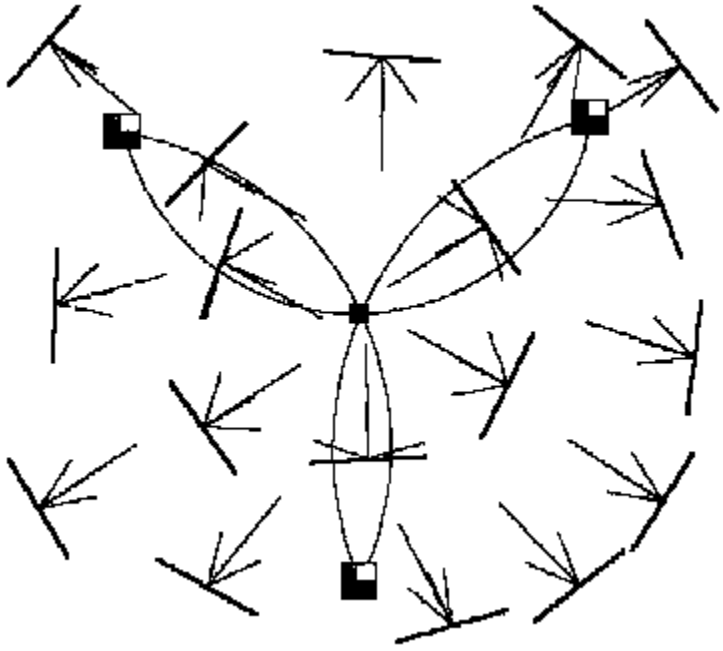

Figure 5. ESim Calculated Headings [Koku, 2003]

In this diagram the three larger boxes are the landmarks, while the small central box is the goal. All of the other marks are ENav calculations at a given point. Each symbol shows a bold line perpendicular to the calculated direction for admissibility purposes, several short lines that show the results of the individual pair wise calculations, and finally the long line that indicates the selected direction. The arcs, which connect the landmarks and goal, are shown for reasons to be discussed later.

ESim also allows the user to simulate the motion of the robot in the scenario (later on, a new program to accomplish similar goals is designed.) This is shown below.

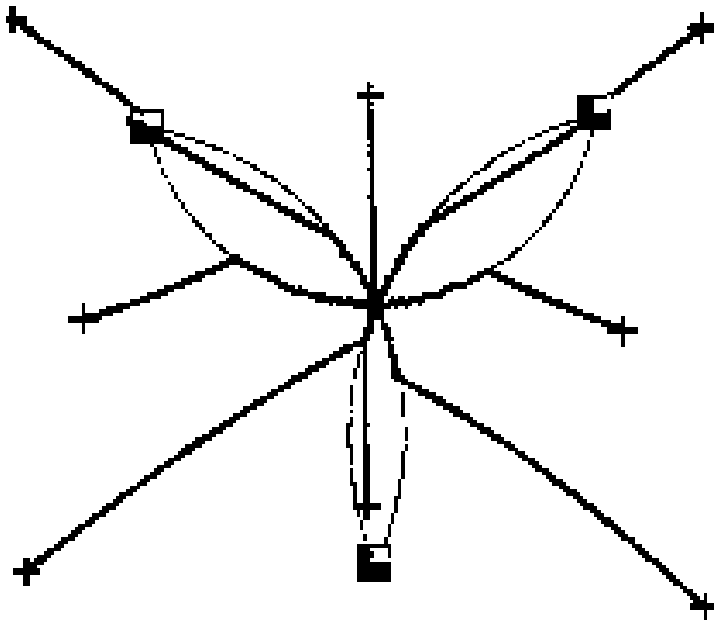

Figure 6. ESim Simulated Paths [Kokum 2003]

In this diagram, the '+' symbols represent the start locations of the robot, and then the simulation traces the robots path toward the goal.

Using ESim, Koku was able to analyze the overall performance of ENav. He did this by considering the simulation's success case by case. These results are reviewed now as they indicated how a 3D ENav could improve upon 2D ENav.

The first case to consider is the two-landmark case. The one landmark case is the special case where ENav simply moves towards the only visual landmark and makes no attempt to localize itself in a goal region. In the two-landmark case, Koku showed that the best ENav can do is to localize the robot on the arc that contains the two landmarks and the goal.

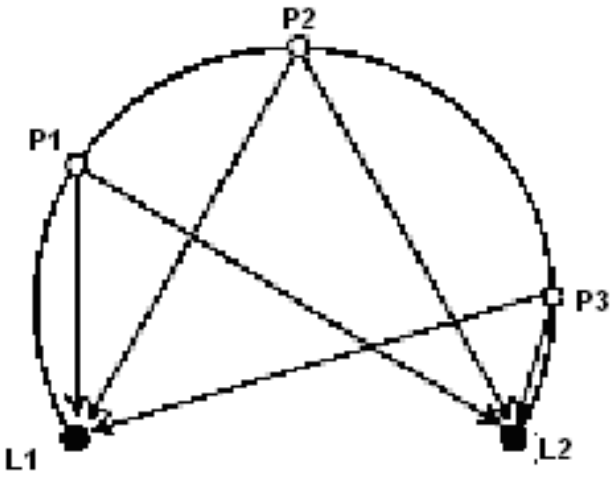

Figure 7. 2D ENav Result for 2 Landmark Case [Koku, 2003]

In the above diagram, assume the true goal region is somewhere on the arc connecting the two landmarks (L1 and L2). It is shown that any point on the arc (P1, P2, P3 etc) will produce the same angle between the landmarks, and have the same ordering of landmarks. For this reason, in the two-landmark case, 2D ENav can only localize the robot to the arc that contains the goal and the two landmarks.

The next case, in which there are three landmarks, is the case Bugra provided the most analysis. This case is more complicated then the previous one however, because the scenario needs to be broken down into regions, as shown below.

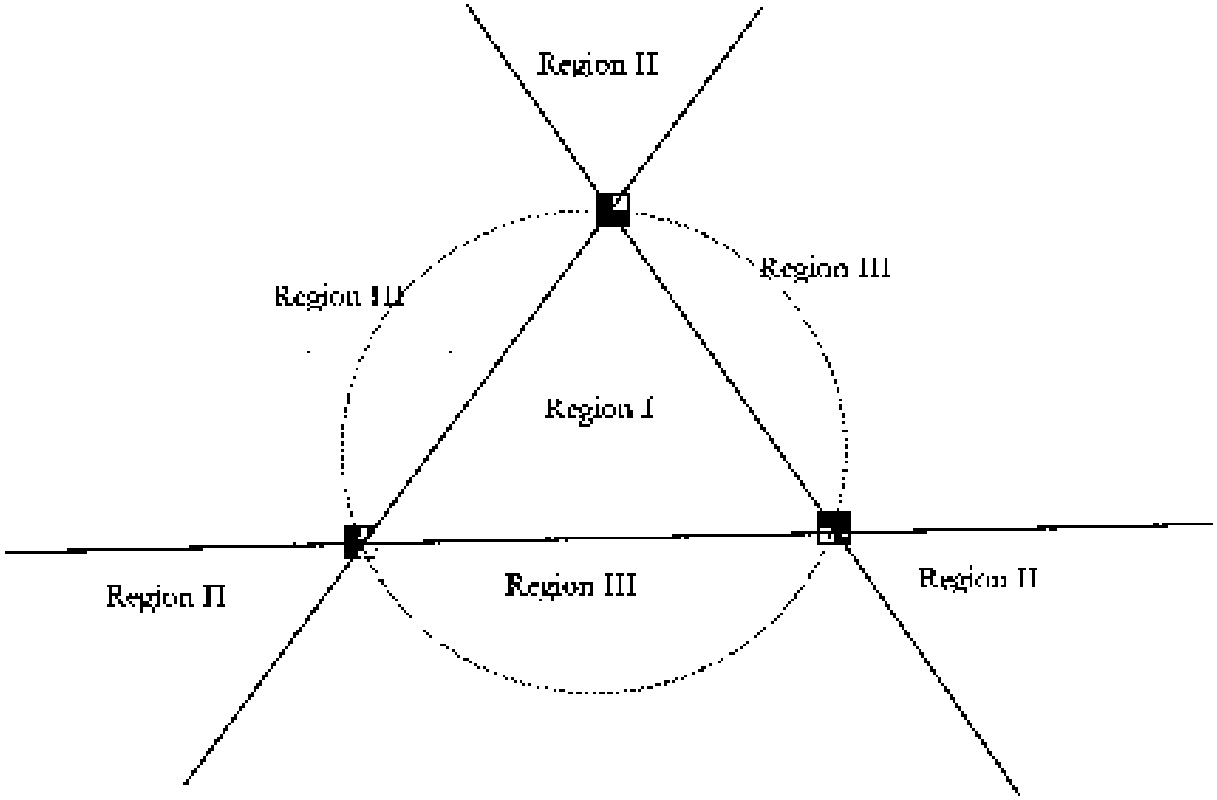

Figure 8. Breaking 3 Landmark Case into Regions [Koku, 2003]

In the above diagram we see that for the three-landmark case there can be three regions of consideration. Depending on which region contains the goal, ENav performs with varying degrees of success.

In the analysis, it is indicated that if the goal is in Region I, the region inside the triangle made by the three landmarks, the robot will always converge on the goal. The same is true for Region II. However, if the goal is in Region III, there is no guarantee that the robot will find the goal. This is shown in the following simulation runs.

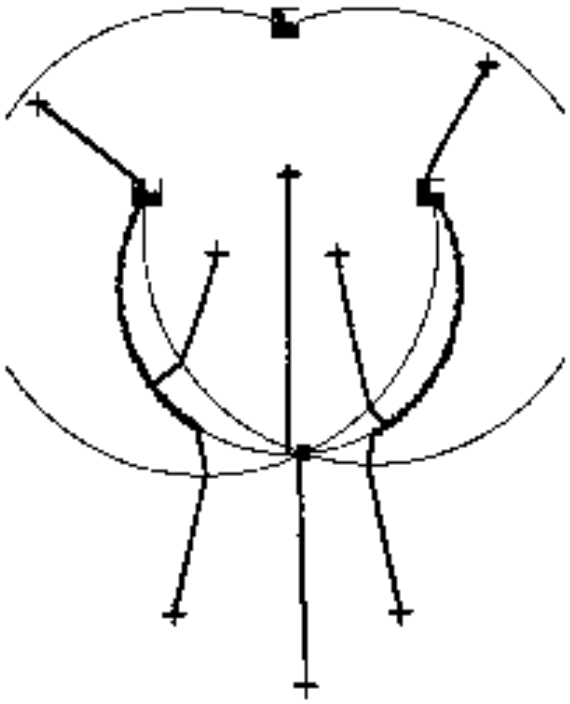

Figure 9. Simulation Results for 3 Landmarks Case with Goal in Region III [Koku, 2003[

In this simulation, we see that for only one of the eight starting points chosen does the robot locate the goal. This problem of non-convergence continues into the 4 Landmark case and up. In these cases, convergence can only be assured if the goal region is within at least one triangle created by any three of the landmarks.

Limitations of 2D ENav

In this section, some of the limitations of 2D ENav are discussed. In later sections is an attempt to demonstrate how work described in this thesis alleviates the majority of these limitations.

The biggest limitation with 2D ENav is that convergence is not guaranteed for 2 or more landmarks. In the 2-landmark case, the robot can at best localize itself on the arc containing the goal and both landmarks. Then in the 3-landmark case, we see that the convergence is dependant on the positioning of the landmarks. This holds as well for the four and plus landmark cases. This is the largest shortcoming of 2D ENav. Koku describes methods for defeating this (using a compass or a map), but solutions are presented later on, which requires no additional hardware or a priori knowledge. [Koku, 2003]

Another deficiency of 2D ENav is that the paths taken by the robot are often "sub-optimal".

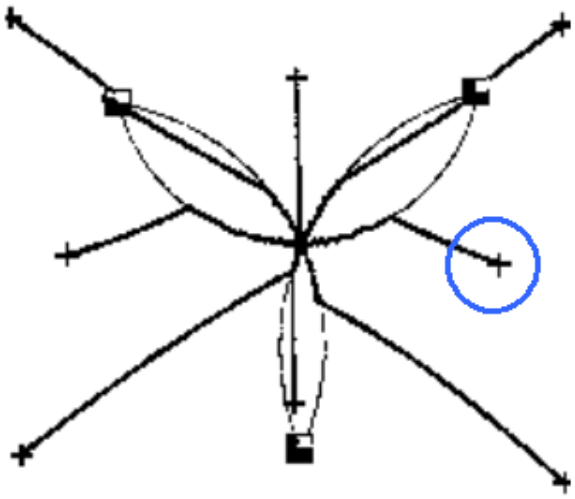

Figure 10. ESim 2D ENav Path Simulations [Koku, 2003[

Consider again the above diagram that demonstrates the path taken by a simulated robot running ENav. If we look at the path that begins at the circled starting point, we see

that the robot moves first to the arc that connects the goal and top 2 landmarks, and then moves along that arc to the goal. This is actually a typical behavior for ENav. Looking again at the following ESim result:

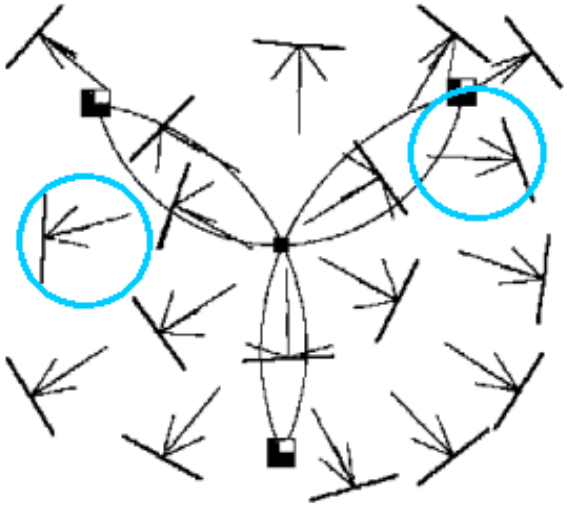

Figure 11. ESim 2D ENav Heading Simulation [Koku, 2003]

We see that the circled directions reflect the ENav behavior of moving first to one of the arcs and then moving along the arcs toward the goal.

In [Pinnete, 1991], the concept of an admissible movement is defined as a movement that reduces the robot's distance to the goal. This would mean any movement that moves the robot further from the goal is inadmissible.

In his analysis, Koku shows that 2D ENav convergent solutions may contain some inadmissible movements. Consider the following example:

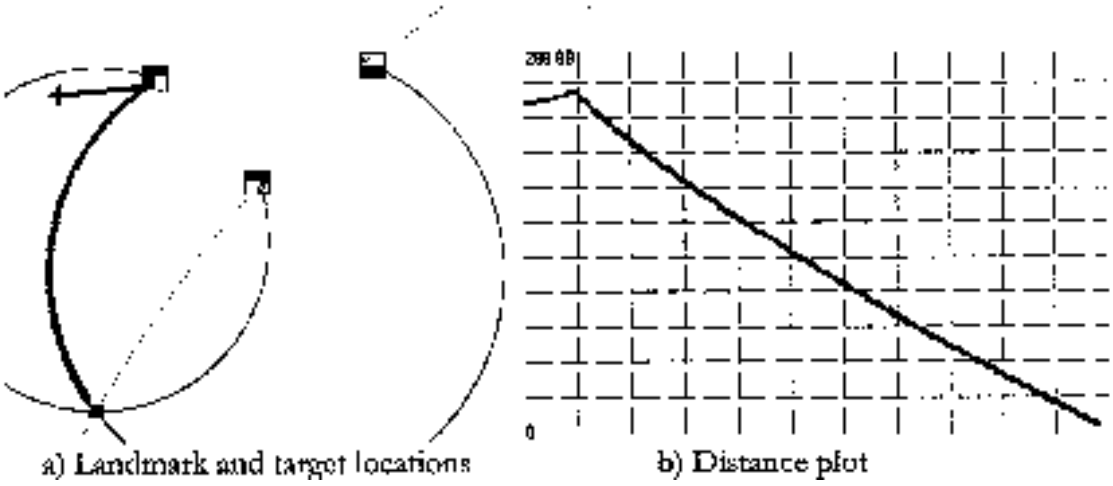

Figure 12. ESim 2D ENav Simulation [Koku, 2003]

In this example we see that the robot at first makes movements that move it away from the landmark, but converges eventually. These moments of inadmissible, or even merely less than direct, motion cause the robot take to longer than necessary to arrive at the goal.

In summary, the limitations of 2D ENav are:

1) Three landmarks are required in order for robot to find goal.
2) Convergence of robot to goal dependent on goal/landmark configuration. (Convergence is not guaranteed)
3) Paths are often non-optimal and inefficient.

CHAPTER III

A 3 DIMENSIONAL EGOCENTRIC NAVIGATIONAL ROBOTIC SYSTEM

Overview

In this section of the paper a design for a 3D ENav based robotic system is designed and discussed. In the previous section, the concepts of an SES, LES, and 2D ENav were discussed and analyzed. A 3D ENav robotic system will improve upon the previously discussed 2D ENav system.

At this point, a complete robotic system, hardware and software, is designed to implement these concepts. This system will be broken down into a number of sub-systems. These include a visual system for recognizing and locating landmarks, a memory system for storing SES's and LES's, a navigational system for deriving a robot heading from an SES and LES, a motor control system for moving the robot based on the navigational heading and finally a controller for coordinating all the systems. Each of these systems will be discussed individually and in the end a complete robotic system for implementing 3D ENav will be designed.

The Visual System

The visual system of the robot is the system charged with recognizing and locating landmarks in the robot's environment. Specifically, the system uses camera and visual recognition to perceive landmarks relative location in relation to the robot for insertion on an SES.

The visual system for the robot can be discussed in two parts: hardware and software. The hardware of the visual system consists of the cameras and related equipment. While the software controls the cameras and equipment, and also processes the information gathered by the cameras. In this section,first the hardware make up of the visual system is discussed. Then a discussion of the software, and finally, the total operation of the system is discussed.

Hardware

The hardware of the visual system consists of six small cameras, a multiplexer and a laptop computer.

The system's cameras are sandwiched between two pieces of orange foam, and are arranged to cover a full 360 area around the robot. Specifically each camera is placed 60 degrees from its neighbor.

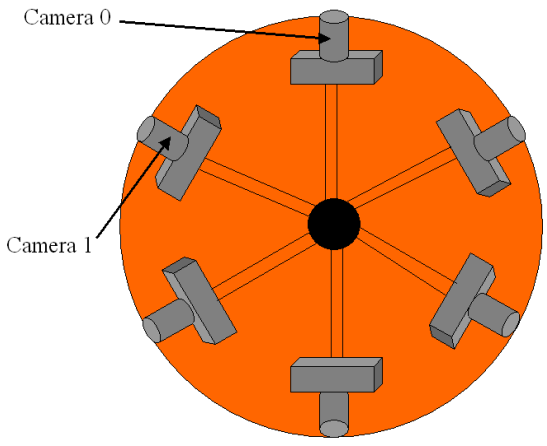

Figure 13. Visual System's Camera Array

The cameras are numbered 0 through 5. Camera 0 is the camera that looks toward the front of the robot, while camera 1 looks toward the front left. 6V DC powers the cameras, and each outputs a standard video signal.

Each of the camera signals is fed into an 8-input multiplexer. The multiplexer can select any one camera to be outputted either by direct setting through contacts, or through ASCII code words entered through a serial port.

The laptop computer is the final piece of hardware that constitutes the visual system. The laptop, through a serial cable controls which camera the multiplexer should output. It also reads in that video signal after an external digitizer converts it. Finally the laptop processes the images and searches for landmarks. The result of this processing is what is returned to the robotic system through a wireless Ethernet connection. This connection set up is shown below.

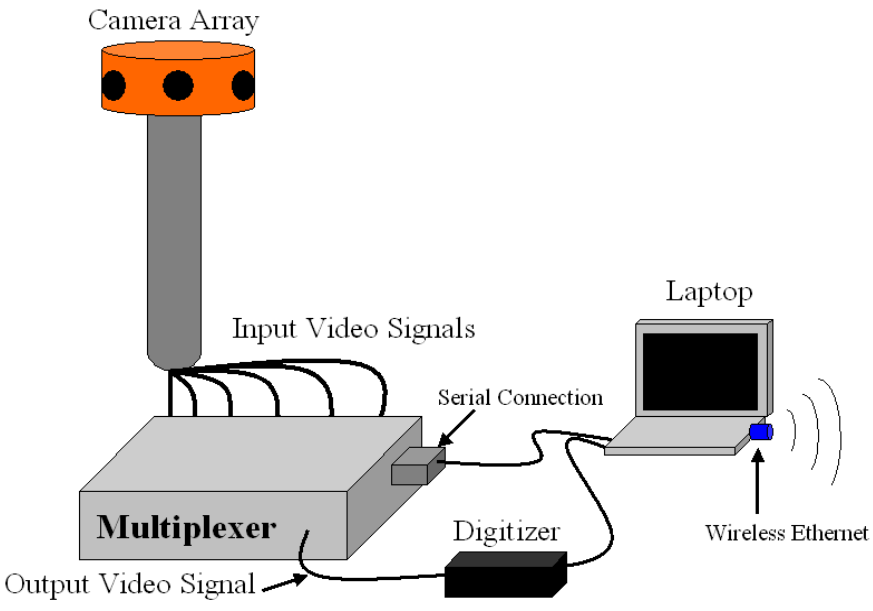

Figure 14. The Visual System Set Up

Software

As discussed earlier, an important part of a robot system using egocentric navigation, is the ability to locate landmarks. This section discusses the algorithm used in my system.

The algorithm has the ability to recognize up to 12 unique landmarks. These landmarks are oak tag cylinders composed of 3 color sections arranged either vertically or horizontally. The three colors that compose the landmark are pink, green and orange. The orientation of the landmark and the order of the colors determine which landmark it is.

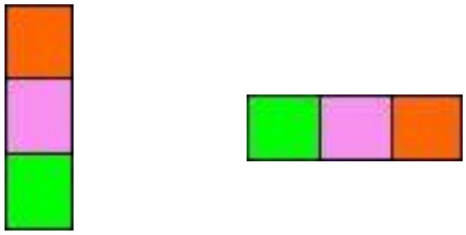

Figure 15. Landmarks

For this research, only the vertical landmarks are used. This decreased the total process time in the landmark detection algorithm while still allowing for enough unique landmarks to test the system. For the remainder of this section only the algorithm that locates vertical landmarks will be discussed, however the horizontal algorithm is basically just a transpose (Will Dodd).

Dr. Wilkes originally conceived the method used to find landmarks within an image. Essentially, the algorithm converts the image into the HSV (Hue, Saturation and Value)

space and then looks for columns with large numbers of pixels in an appropriate Hue space, and high values of Saturation and Value. This takes advantage of the face that the landmarks are composed of bright, saturated ("pure") colors. If there is a group of columns together, where each column contains large numbers of acceptable pixels, these columns probably contain a landmark.

To demonstrate look at the example below:

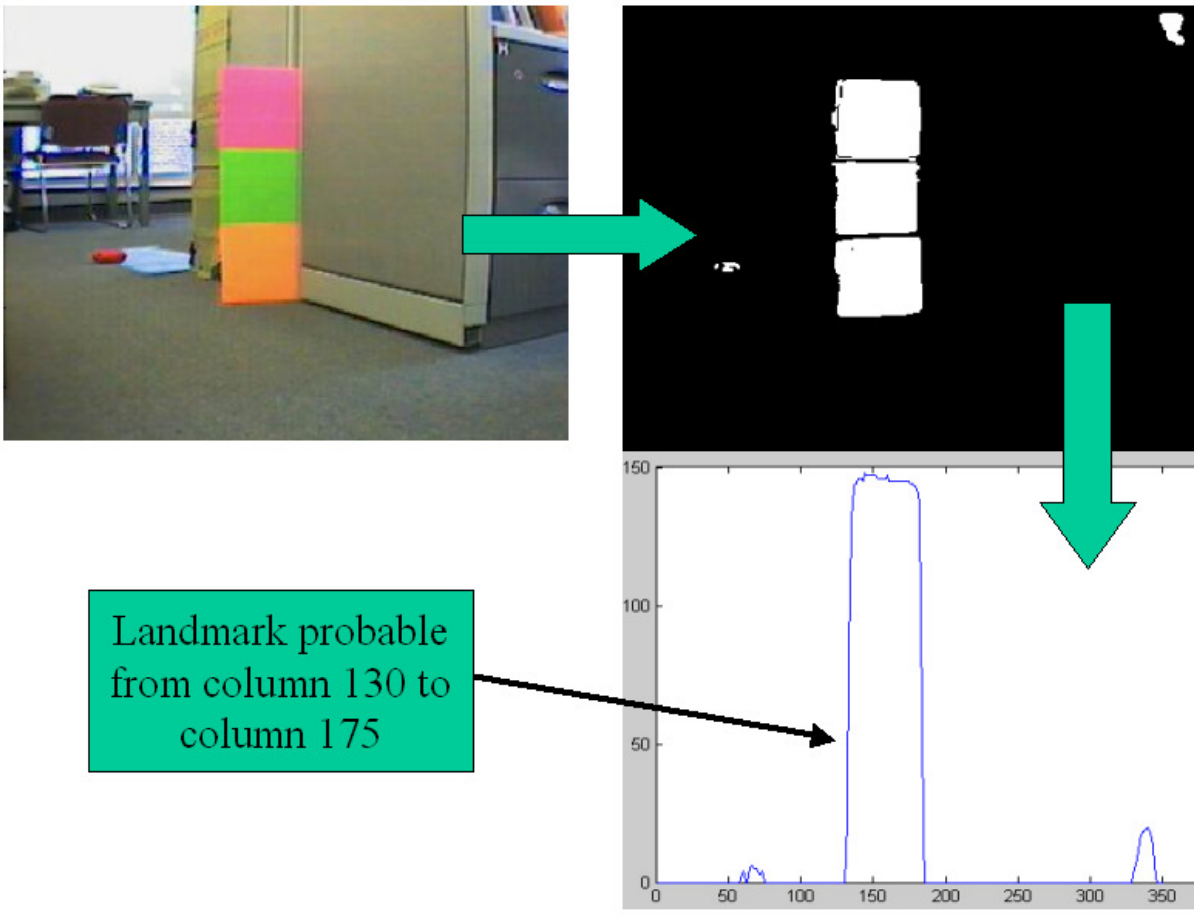

Figure 16. Example of Finding Landmark Region

In the first step every pixel is converted into the HSV space. Next the algorithm finds all the pixels where the product of the saturation and value are greater then predetermined threshold value and whose hue is in range of one of the three colors on the landmark. From this point on such pixels will be referred to as "acceptable pixels". This produces a map of Boolean values displayed in the top right. The next step is to "sum down" all the columns of the Boolean map, which results in a vector of sums, which is plotted in the bottom left. Finally the algorithm considers the sum vector and selects any regions with large sums as being probable landmarks. This leads to the conclusion that there is a landmark from column 130 to 175. In effect, it designates the region from column 130 to 175 as a "landmark region", or a region likely to contain a landmark.

This algorithm is very useful because it takes advantage of the characteristics of the selected landmarks in order to locate them in an image without excessive computation. However it is at this point incomplete. The algorithm now needs to both identify the landmark (what order do the colors appear), and assign x and y locations for the landmark.

For location, the centroid of the landmark is selected as the location of the landmark. In order to solve for this centroid, a number of methods are possible, however mean and median are the most obvious. The first method would be to average all row and column values for all "acceptable pixels" within the landmark region decided in the preceding step. However, this method proved to be too susceptible to "outliers", or the low number of pixels that differ largely from the mean. What this means is that if any pixels in the "landmark region" were deemed "acceptable" but were in fact not part of the actual landmark they would have an undesirably large effect on determining the centroid of the landmark. For this

reason, the median of all the "acceptable" pixels was chosen due to its resistance to outliers, it produces a more consistent estimate of the centroid of the landmark.

Additionally, the algorithm needs to identify the landmark. One method to accomplish this is to determine the centroids of each of the three colors in the region separately, and then order the row-values of these centroids to classify the landmark. This is demonstrated below.

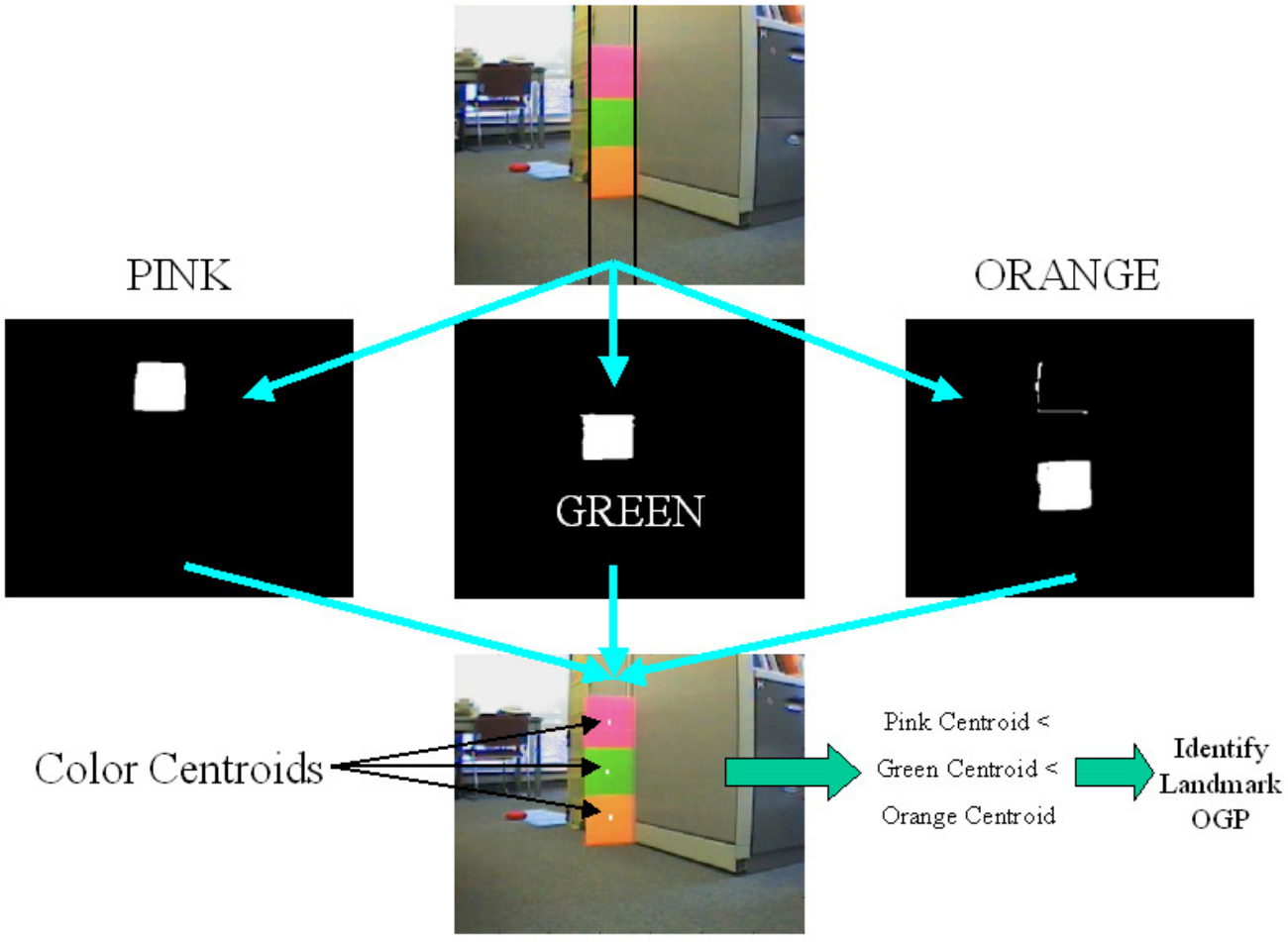

Figure 17. Identifying a Landmark

At this point, the region containing the landmark has been found by the algorithm. It then proceeds to locate all the pixels, within the region, for each of the appropriate hues that have a saturation and value product above some threshold ("acceptable" for this color). For each color, it builds a list of the row values of all the acceptable pixels, maintaining an ascending order to the list. When all the acceptable pixels for this region and color have been counted, it selects the center most row value as the centroid. Note that for this demonstration, this process has been done for the columns as well, but in actual use is not because the height is the only concern.

Notice that the orange map has noise where the edge of the pink square appears orange. This demonstrates the usefulness of using the median method. These "outliers" could move the centroid up a bit had averaging been used, but the much larger amount of "acceptable" pixels in the correct square ensure that the median is close to the desired, or true, location.

Finding the median may seem bit overkill for simply trying to identify the order of the color squares on the landmark. However, because the median values will be useful for later calculations, it is useful to calculate and use them now.

Also, at this point there is an issue of arbitrary value selection. What hue ranges define orange? What should the threshold be in the saturation times value calculation? These questions will be approached later on, but for the values are selected based on that they were suggested by Dr. Wilkes and were intuitive.

The total centroid for the entire landmark can now be calculated in the same way as the individual centroids for each color. We simply find the median values for all the rows and columns of all the pixels, in any of the 3 hue spaces.

With this we have a working algorithm for locating our landmarks within an image. However, it suffers from a number of problems. Firstly, if two landmarks are next to each other, or overlapping, the part of the algorithm that defines the region will see one large landmark, instead of two distinct landmarks. The resulting classification will be unpredictable, as will the decided location.

Additionally, it is possible for a bright colored region in the robots environment to contain enough "acceptable" pixels to cause the algorithm to designate it a landmark region. Again, the location and classification will be unpredictable.

These two problems are serious. If either were to occur, it would cause the robot to "hallucinate" (Dr. Wilkes), and see landmarks where it shouldn't. Or else assign them to the wrong location. Either of these errors will cause ENav to malfunction.

A final problem with the algorithm, which is less serious then the above two, is that it doesn't take advantage of all the information it has about the characteristics of the landmarks. This leads to system to process longer then is really necessary. This is a lesser problem then "hallucinating" landmarks, but inefficiency is something to be avoided if possible.

This leads to the improvements made to the vision algorithm. It may be helpful at this point to review in brief the algorithm as it is now.

The figure below briefly demonstrates the workings of the algorithm. The algorithm first decides upon the landmark regions, using the sums of the columns method. It then simply classifies each region and derives a centroid.

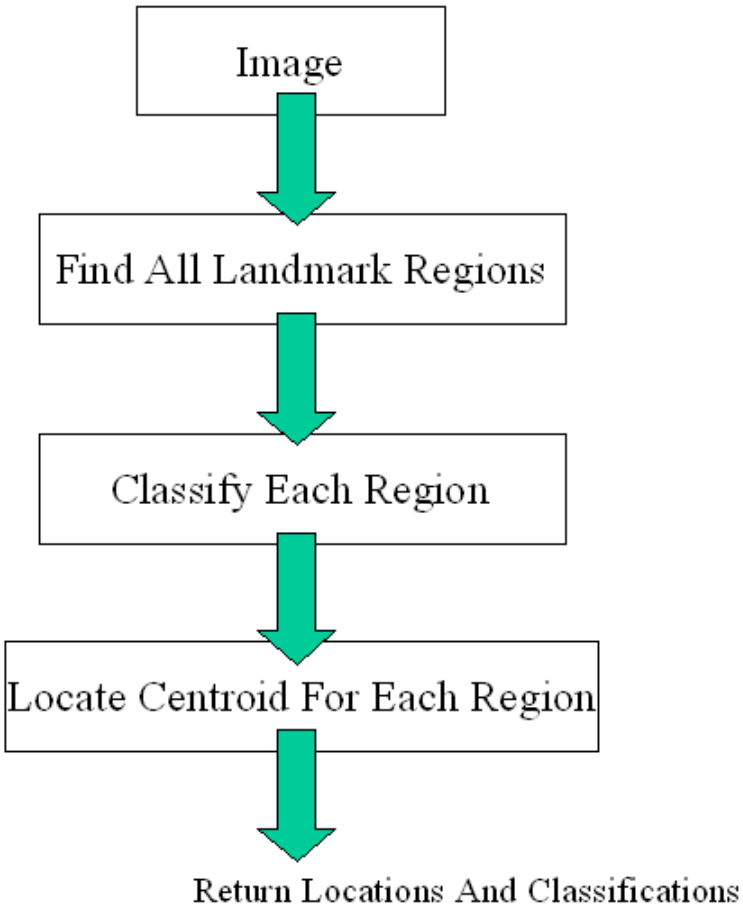

Figure 18. Original Landmark Recognition Algorithm

The first improvement that can be made is to introduce a step after finding all the landmark regions. This step would attempt to decide whether there is more than one landmark. The method employed to decide this as follows. First, for each column in the region, determine the centroid for each color (using the median). After this is done, move through the columns from left to right. If for any color, the median jumps from one value to another, break the region in two at the point.

This decision has to be qualified however by two points. Any jumps near the edge of the region are not considered because the edges are very noisy due to shadowing, etc. Also, if the median level jumps to a new value, but then jumps again, then the jump is probably just noise, and does not represent a genuine new landmark.

To demonstrate consider the image below,

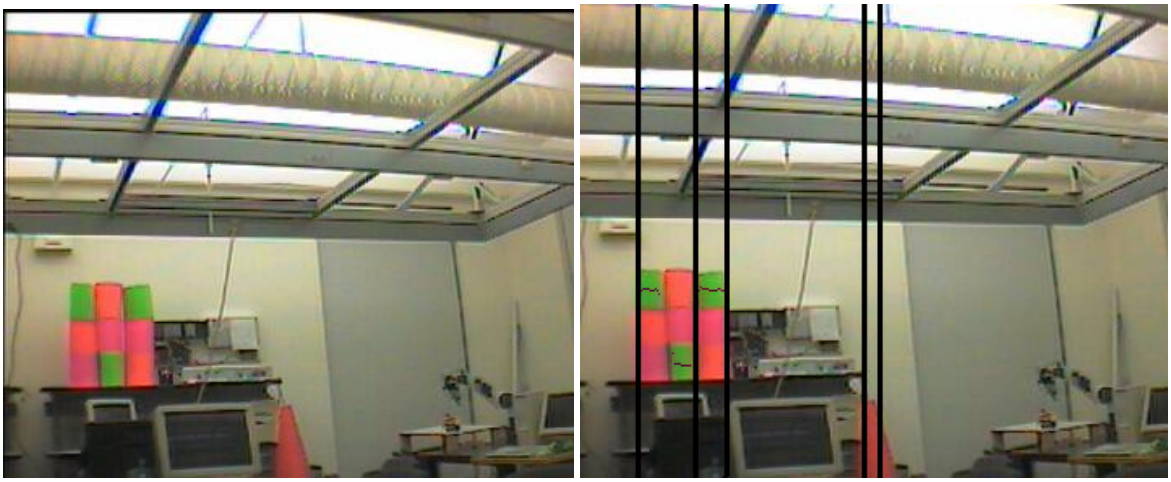

Figure 19. Image for Processing

This image on the left contains 3 unique landmarks. However, the 3 landmarks are directly next to each other, so probably, the algorithm will assign all 3 to one landmark region.

On the right we see the actual results of the algorithms decision on landmark regions. It has incorrectly grouped the left two into one region. Notice though, that for the green color, the centroids of each row have been marked. Now the algorithm, upon evaluating

from left to right, will find the centroid jump, and in such a way that the new centroid level is maintained after the jump. It therefore breaks the region in two at the jump point.

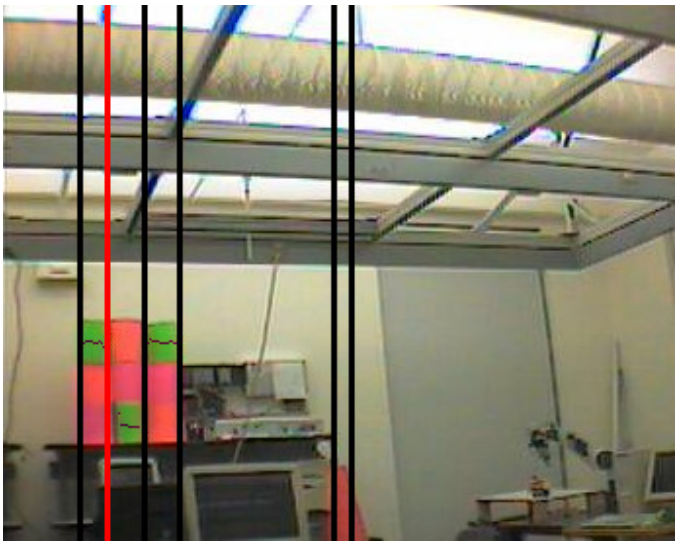

Figure 20. Breaking the Landmark Region

This process works well in discovering two landmarks classified to one region. The next step in the process should be to classify each region. However we notice another problem. The traffic cone is composed of "acceptable" orange pixels. It is also tall enough so that the sums for some columns are large enough to have it designated a landmark region. However, it is not a landmark, and if it is classified as one, it will cause the system to report a landmark where there isn't one.

The solution to this problem is as follows. First we find the medians for each color, for each region, of the column medians. This might look as follows:

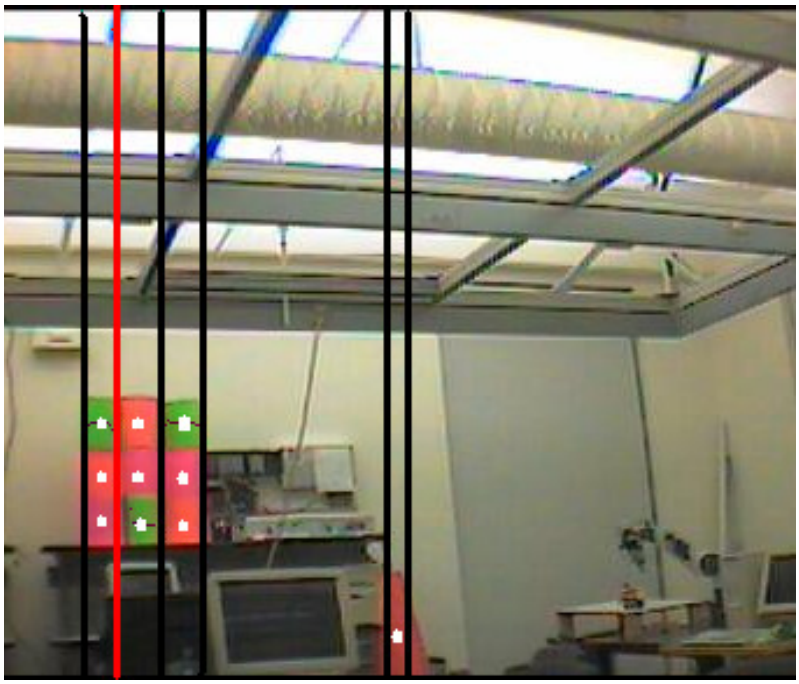

Figure 21. Median for Each Color

Now, we know that for a true landmark, there should be three median values (one for each color), and that these medians should be equally spaced apart. On our false landmark, we see that there are only orange pixels, and this produces only an orange median. It therefore makes sense, to eliminate all regions that do not contain a median color for each landmark. Additionally, suppose by chance there was a pink neon sign above the cone, and a green one above that. This fluke could also be eliminated on the basis that the medians are not equally spaced.

An additional advantage of the last computation is that the medians computed can be used for further computing. First, after we eliminate all none valid landmark regions, we can use the computed medians in the valid regions to classify the landmark. We can also use them to define an upper and lower bound for the landmark. So say the difference between

the upper two medians is Δ_1, and between the bottom two is Δ_2. We already know these two deltas are similar, but we compute their average (Δ), and then use the estimate that the top of the landmark is half a Δ from the top median, and half a Δ below the bottom median. These values (the top and bottom rows of the landmark) are useful in that say there was a neon sign above the landmark that contained a number of "acceptable" pink pixels. Ordinarily these will have an effect (even if small) on the computation of the total centroid, which is the next step. However, now when we compute the total centroid we only consider pixels within the region that are above the bottom row, and below the top. This adds efficiency and effectiveness.

Now the algorithm is complete, here are the steps in total for the algorithm

1) Find all "acceptable" pixels for each color, Boolean sum of these give the total
2) Sum columns to get the sum vector and define landmark regions
3) Find all color medians for each column and break regions wherever medians jump appropriately
4) Find total medians for each color, eliminate invalid regions and classify
5) Find top and bottom rows of landmark and determine total centroids, return

With this the landmark recognition algorithm is complete. However it still needs to be "tuned". There are a certain number of values used throughout the algorithm that are essentially arbitrary. Specifically these are:

1) The hue ranges that define each color
2) The threshold value that saturation times value must exceed for "acceptability"
3) The height of the sum in the first step which indicates a landmark region
4) The minimum width of a landmark region (to eliminate single blips)

5) How much change in color median indicates a break

In order to determine good values, tests were run on the algorithm in various lighting and settings to find values that performed best. To do this a test program was written. The algorithm itself is entirely contained in a C++ class called Vis_data. This class can load an image, and perform all necessary functions to locate the landmarks. It additionally allows outside functions to adjust the parameters of the search (the values listed above). The program just uses the class, and allows the user to adjust parameters to find the best set, as well as save and load parameters for use over time. Additionally, the program works with another program for grabbing "scenes" or series of six images which would be what the actual robot will work with. This allowed me to discover the best parameter values to use. Below is an example of this program in use.

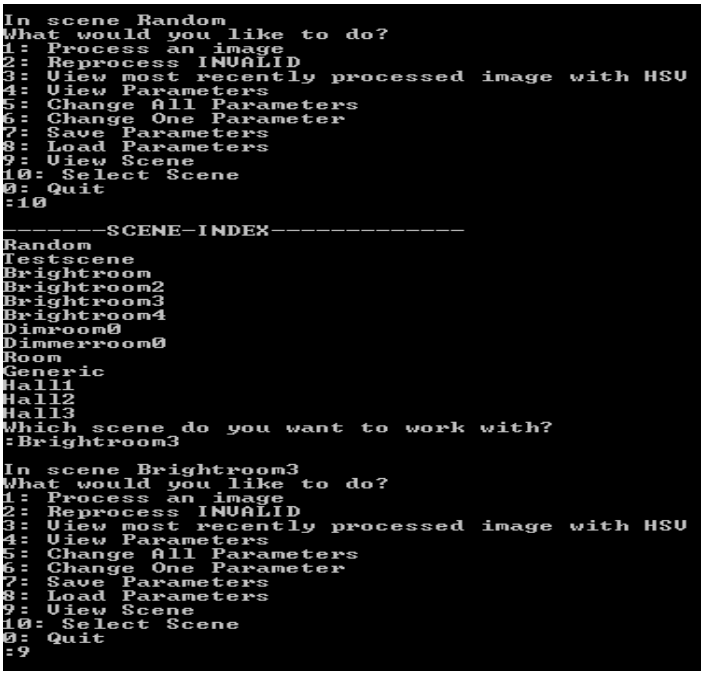

Figure 22. Screenshot

First the user selects a scene to work with. In this case the user selects Brightroom3, and then asks the program to view it.

Figure 23. The "Brightroom3" Scene

These are the six images taken by the robot of the room around it. Now the user can process each image as he chooses to try to find good values for the parameters.

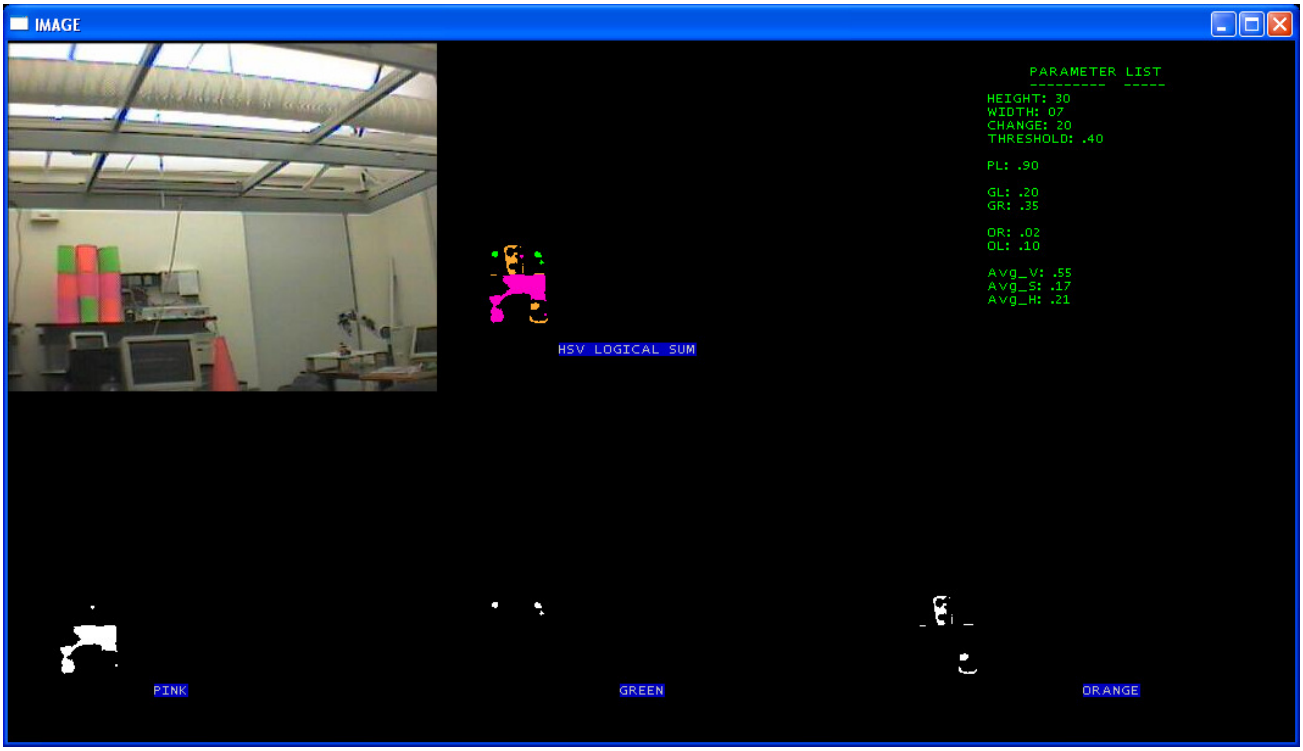

Figure 24. Processing an Image from the Scene

This is the display after processing the fifth image. We see that the program has found no landmarks and this is due to fact the algorithm seems to be too strict in its allocating pixels as acceptable. Now, we load a set of parameters that have proved effective.

```
-------INDEX-------------
Original
Orangeadjust
Expandedorange
Orangelow
Open
Opener
Testing
Good

Please enter which set you would like to load
:Good
Parameters for Good

Parameter Set Up:

Height  Width   Change  Threshold       PL      GL      GR      OL      OR
------  -----   ------  ---------       --      --      --      --      --
15      8       20      0.25            0.9     0.2     0.35    0       0.06
```

Figure 25. Loading Parameters

Here the user selects to load a parameter set and selects the set titled “Good.” This set is displayed at the bottom. Height is the minimum number of “acceptable” pixels in a column necessary to signal a landmark region. Width is the minimum width (in pixels) of the region for it to be considered value. Change is the number of pixels a jump within a landmark region needs to be to result in the region being broken. Threshold is the value that the product of saturation and value must exceed in a pixel for that pixel to be “acceptable”. And the remaining values are the boundaries of the hue spaces for the three colors (PL means “Pink Left” and so on). Now, reprocessing the image shows

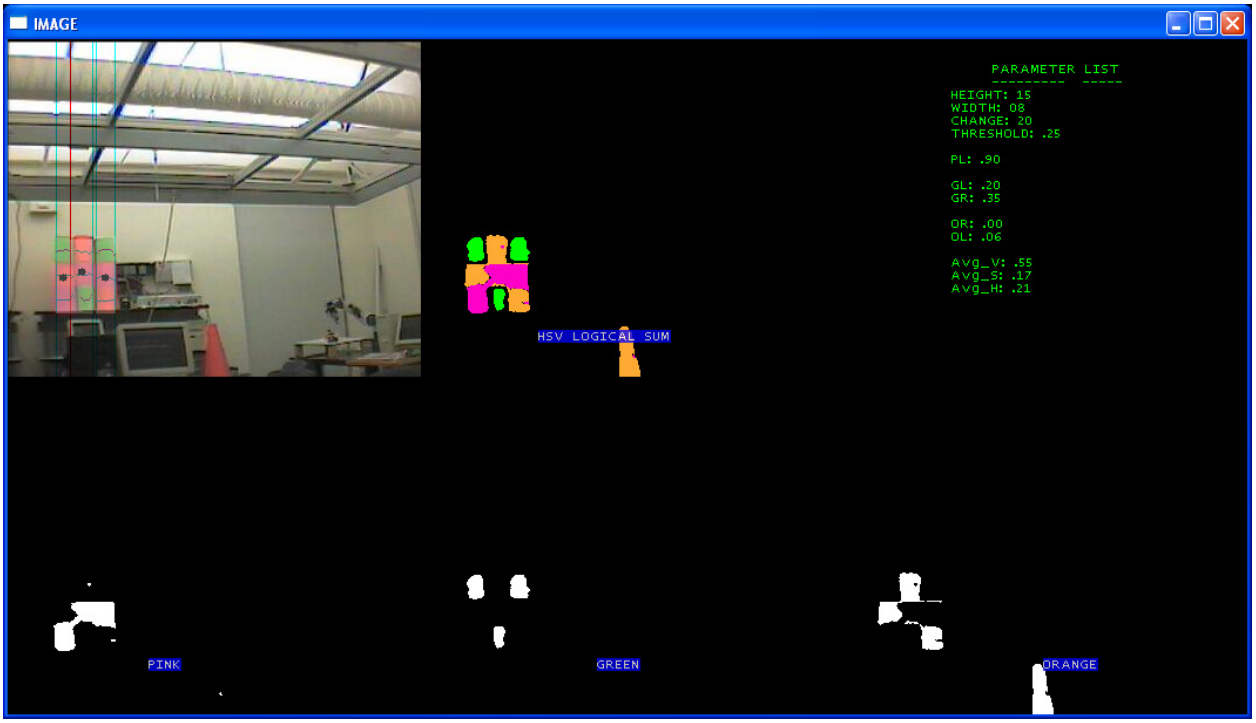

Figure 26. Reprocessing the Image with Better Parameters

This is a much better result. The landmarks are located and classified correctly. We notice that although the orange cone is in the scene, it is not classified a landmark.

After testing a number of scenes, the parameter set discovered to work best is the set used above. Essentially, it is initially as permissive in what it deems an "acceptable" pixel and landmark region and then relies on the later checks to remove all the invalid regions. This has proven effective at finding most landmarks while not producing any "hallucinations".

Overall Visual System Operation

Now that the algorithm to locate and classify landmarks within an image is established, we must finalize the workings of the overall visual system.

We can start by defining a scan as a single operation of the visual system. Specifically, a scan will be when the visual system checks all the cameras for landmarks and then reports them back to the overall robotic system. This will be the fundamental operation of the visual system on the robot.

A scan will commence when it is requested by the overall robotic system. It terminates when it returns the locations of all the landmarks that it found. In between it must grab a frame from each of the cameras and perform the landmark recognition algorithm on each of those images. One way to do this would be to first grab a frame from each camera, and then process all the images. However, a more efficient way is to process each image, immediately after the computer requests a new camera from the multiplexer. This is because the digitizer requires time to adjust to the new input, and so rather that having the laptop wait

idly, it makes sense to process the image grabbed from the previous camera concurrently with this adjustment. In practice, a scan (from start to finish) takes about 8 to 10 seconds.

The final consideration is in what should the locations of the found landmarks be returned. The method I've selected is order all the possible landmarks in the following way PGO (pink green orange), POG, GPO, GOP, OPG, OGP, and therefore number them landmarks 0 through 5. This is the convention used throughout the robot. Now we returned an array which is 18 integers long which contains landmark 0's x-value, y-value and the number of the camera which it is in, follow by landmark 1's x-value and so on. If a landmark was not found on any camera, the number –1 is used for all three values of that landmark to flag that it was not found. This array I've called a "landmark array".
Here is a sample landmark array:

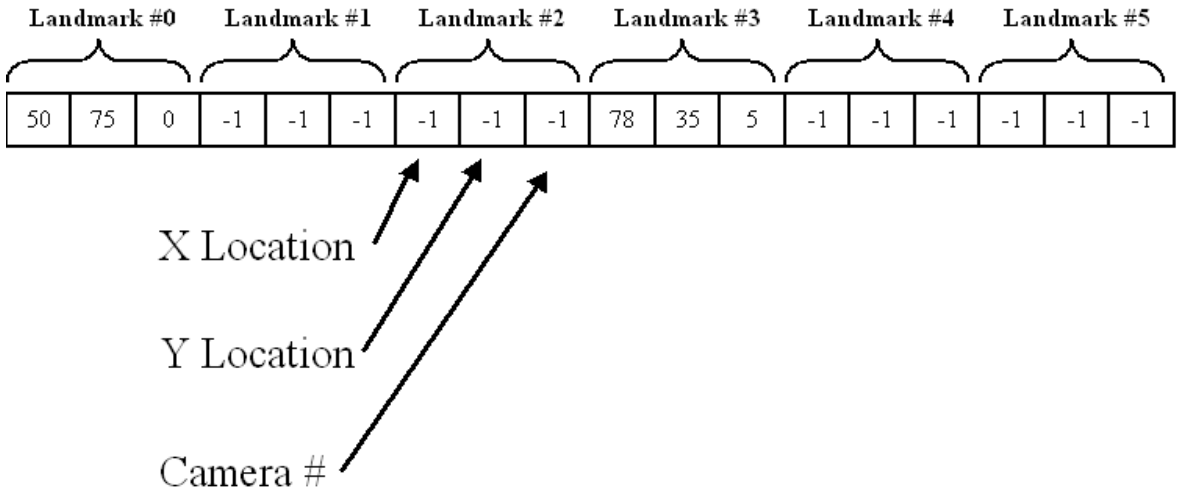

Figure 27. Sample Landmark Array

In this case, the scan concludes that Landmark #0 (PGO) is on camera 0 at the X location 50, the Y location 75. Landmark #3 (GOP) is on camera 5 at X location 75 and Y location 38. The X and Y locations are the pixel locations of the centroid on the image, which is set up like this:

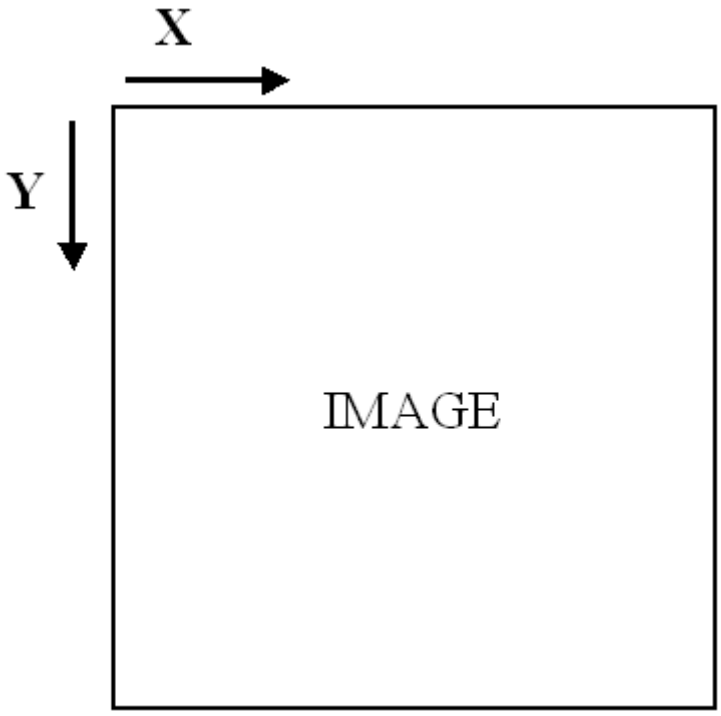

Figure 28. X and Y Locations on an Image

To summarize then, the basic block of the visual system could be seen as a sub-system that receives a signal to initialize a scan, and then returns a "landmark array".

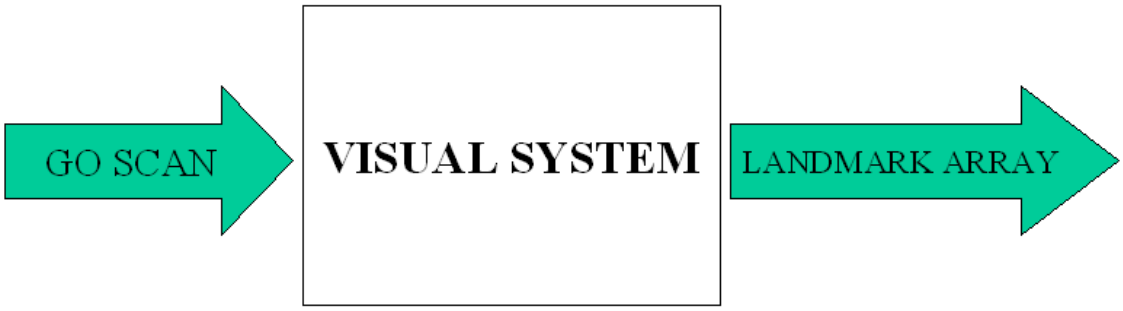

Figure 29. Visual System Overall Behavior

The landmark array returned by the visual system must now be converted into the memory structure used by the robot, the SES. This is done within the next subsystem, the memory subsystem.

The Memory System

In the last section, we reviewed one component of the overall robotic system, the visual system. This system, when told to do so, scans the room and returns the locations of all the landmarks in the form of a "landmark" array. The memory subsystem, must then convert this into the form used by egocentric navigation, the Sensory Egosphere (SES), and store the SES. In addition to converting visual data into an SES, the memory subsystem also stores Landmark Egospheres (LES) for use in egocentric navigation. Finally, the memory subsystem must also post the egospheres to a database that is consistent with the structure of egosphere storage used on other CIS robots, such as ISAC. This is so that any robot can access this robot's SES in the standard form. The main functionality of the memory subsystem then is to:

1) Build an SES out of visual data, and store in memory

2) Store LES's for use with ENav

3) Return on demand the robot's SES or any stored LES

4) Post SES to networked database

5) Visualize SES for testing and display purposes

First, we review the hardware affiliated with the memory subsystem.

Hardware

The memory subsystem consists of two computers, both off the robot. The first computer is the central control for the robot, this computer will also be responsible for calculating navigation, and so is the main computer. The other computer is a server that contains the formatted database program and visualization tools.

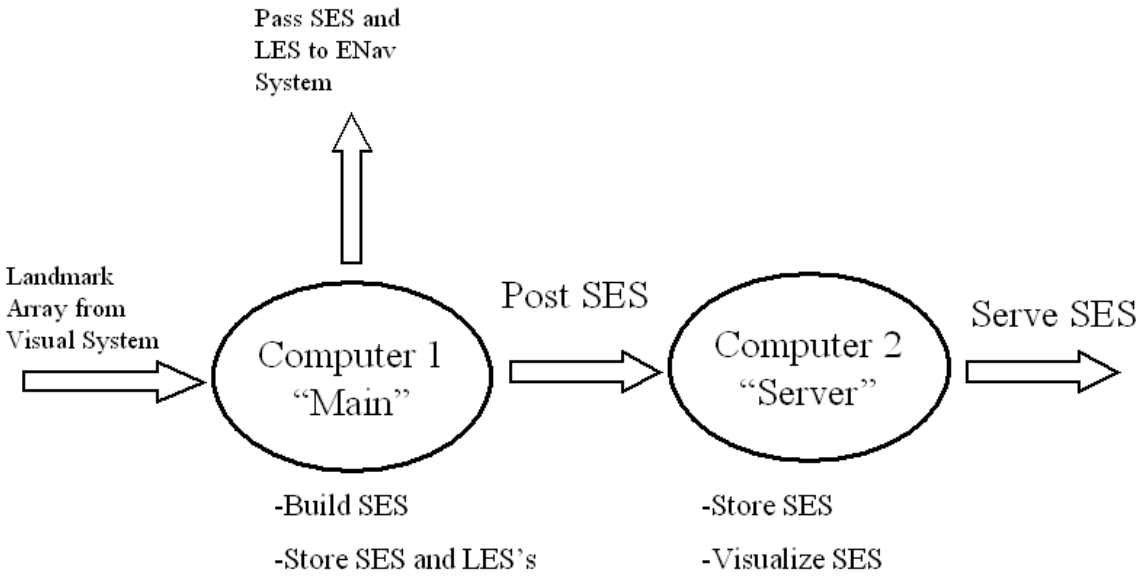

Figure 30. Chart of Memory System

This dichotomy leads to the idea that the first computer is local to the robot, and through the second computer it forms the bridge to other robots and to users. We can now turn to functionality of the main computer and then the "Server" later on.

Software -- Memory on the Main Computer

In the background section of the thesis, we reviewed the concept of egospheres. Specifically we arrived at the concepts of the Sensory Egosphere and Landmark Egosphere as memory structures for storing information about the robots current or desired perception of the world. In the discussion of egocentric navigation, these egospheres were reduced to "2D". For this robot, we will be using the full 3D egosphere, 3D in reference to the 3-Dimensional Sphere, vs. a 2-Dimensional circle.

Because the system is using 3D space, the coordinates of landmarks on the egosphere must be stored using two angles: azimuth and elevation. The angles are established as follows: the azimuth measures the angle along the floor, while the elevation measures the angle off the floor. The azimuth angle is zero directly in front of the robot and increases as you go clockwise around the base of the egosphere.

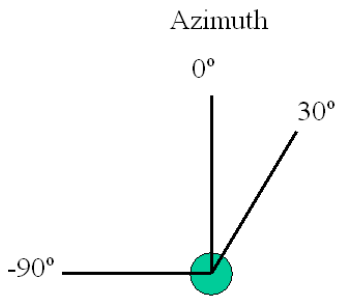

Figure 31. The Azimuth Angle

If the top of the page is forward for the robot, then it is shown that directly ahead is 0 degrees while something on the left is at –90 degrees.

For the elevation angle, the horizon is 0 degrees and directly up is +90 degrees.

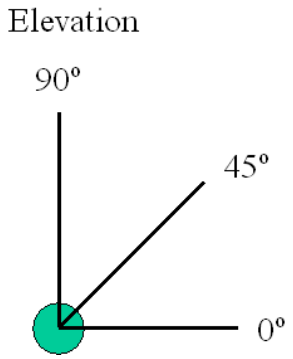

Figure 32. The Elevation Angle

For the version of the SES used on the robot, both angles are measured in radians. The azimuth angle can be represented as ranging from 0 to 2 Pi or else –Pi to Pi. The elevation angle is limited to 0 to Pi / 2.

Now that the angular indexes for the robot are established, the general structure for the SES and LES are described. In the main computer, both the SES and LES are stored in a specialized structure called an egosphere in the code (on the server computer, the SES is stored in a specially generated MySQL database generated through Kim Hambuchen's

Visual Basic SES code). [Hambuchen 2004] The egosphere structure for the robot is set up as shown:

Local Egosphere Structure

Landmark	#0	#1	#2	#3	#4	#5
Azimuth	-.1	1000	1000	1000	.4	1000
Elevation	.5	1000	1000	1000	.2	1000

Figure 33. Egosphere Structure

In this set up we see that the egosphere stores for each landmark the Azimuth and Elevation angles. If a landmark is not present in the egosphere, it stores 1000, which is an invalid angle. In the cases where the landmark is present in the egosphere, it stores the angles in radian form.

The memory subsystem contains to types of egospheres: the SES and the LES. For the SES, it has a single egosphere. The SES is a representation of the current state of the world around the robot and is therefore directly related to the output of the visual system (more about this later). The system also contains an array of egospheres to represent multiple LES's. Each LES represents some goal location in the form of where the landmarks should be in the SES if the robot were at that location.

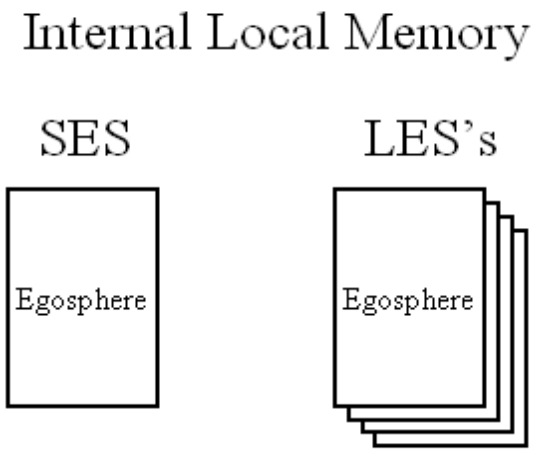

Figure 34. Internal Memory Setup

In the last section about the visual system, it was shown that the process carried out by the visual system terminates when it passes the "Landmark Array" to the controlling system. It is now the function of the memory system to convert that raw sensory data into the SES for the robot.

To do this, for each landmark in the "Landmark Array", the X location and camera number information are combined to determine the Azimuth angle, while the Y location is used to find the Elevation angle.

Overlapping the description of the camera array from the visual system with the Azimuth angle display from this section shows the following:

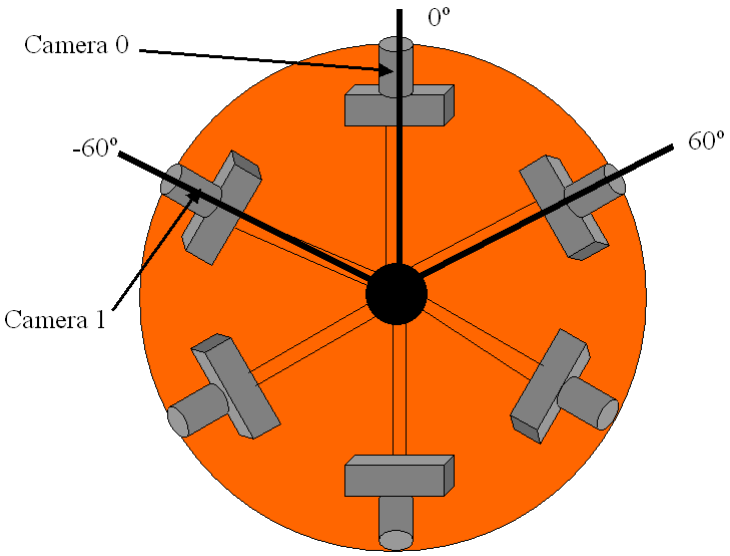

Figure 35. Azimuth Axis Overlaid on Camera Array

If for instance a landmark in the array returns an X location in the center of the image, and is on camera 0, then the Azimuth angle for the landmark is 0. If however it is centered on camera 1, it is at –60 degrees (-PI/3), if it is on the right side of the image from camera 1 then its angle may be –45 degrees, and so on.

Calculation for the elevation angle is more straightforward. Regardless of the camera, the Y location is used to directly compute the elevation of the landmark.

Using these rules, the SES for the robot is computed directly from the "Landmark Array" returned from the visual system. An example of this is shown below:

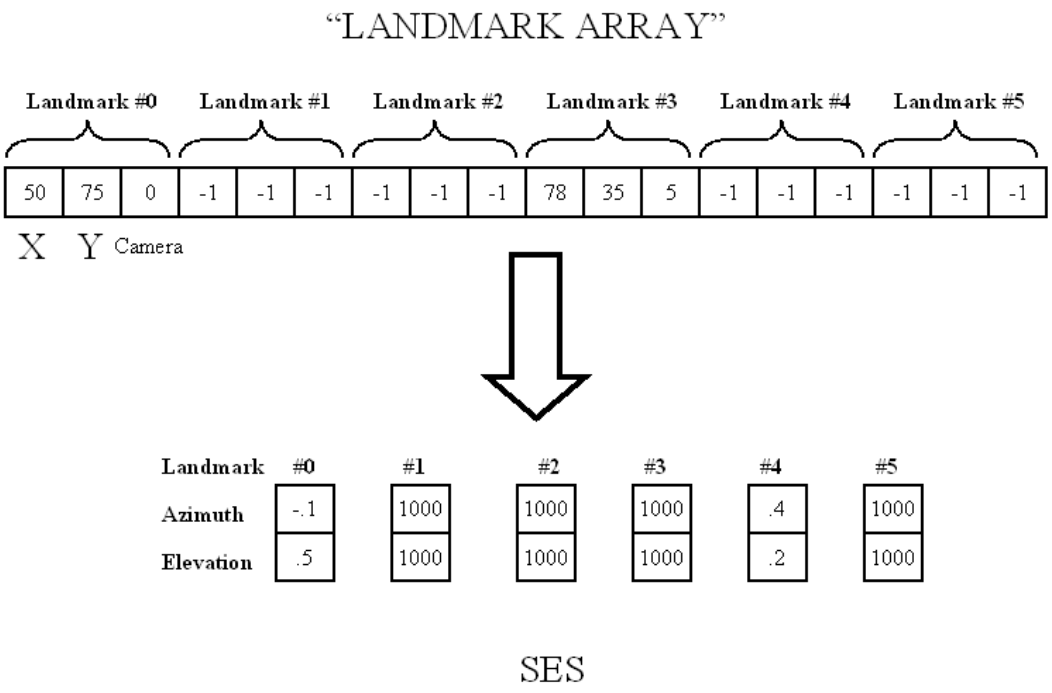

Figure 36. Converting Landmark Array into SES

In this case, the visual returns that it found two landmarks, one on camera 0, and one on camera 5. The SES is computed accordingly.

The stored LES's are locations. They can come from either direct user creation, as in a user manually typing in the angles of an LES. They could also be a stored SES from this robot, or even another robot. For instance, if we could put the robot in some goal region, have it grab an SES, then store that SES as an LES. Now the robot can use that LES to find its way back to the goal region.

Software -- Server Computer

As mentioned earlier the function of the Server computer is to hold the SES of the robot in the MySQL database used by other robots in the CIS. The robot's local computer posts landmark locations to the database through Vanderbilt's Wireless Ethernet.

One note about this part is that because it is not essential to testing a single robot, the server computer is not fully implemented, although a database has been created, as well as an adapted visualization program.

The Navigation System

3-Dimensional Egospheric Navigation

In the previous sections, it is shown how the visual system locates landmarks in the world, and how the memory system converts this into an SES and also stores this SES along with an array of LES's. At this point the navigational system of the robot can operate on the SES and an LES supplied by the memory system to determine an appropriate direction for the robot to travel in.

To do this, the robot uses 3 Dimensional Egospheric Navigation. This is a method that was developed for this thesis, which is based on the work done in [A] and [D], and received much advice and help from Dr. Wilkes.

In his thesis, Bugra suggests that his method of ENav could be expanded to 3 dimensions with no loss of generality in the case that robot itself can move in three dimensions, such as unmanned undersea vehicles (UUV). [Free Dictionary Dot Com] A system was designed however, which uses the 3D egospheres to control a robot that can only move in 2 dimensions.

In this first part of this section, the theory, analysis, simulations and advantages of this new egospheric navigation algorithm are discussed. In the second part, the actual implementation of the system on the robot in terms of hardware and software is discussed.

Brief Review of 2D ENav Concepts and Analysis

First, it would be helpful to briefly review the concepts of 2D ENav. In this version, first the SES and LES are reduced to 2D. Next, all the landmarks that occur in both egospheres are compared pair-wise. Specifically, each the angles between each landmark pair are contrasted between SES and LES, and the robot is assigned a direction along the bisector of the angle, either towards or away from the landmarks, depending on the result of the angular comparison.

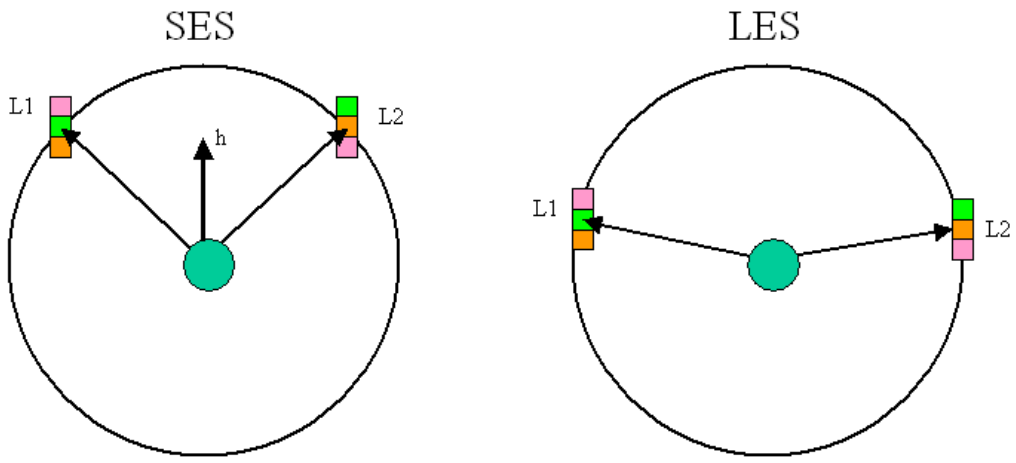

Figure 37. Calculating 2D ENav

In the final step, the resultant direction from each pair is computed to determine a final heading. This leads to the vector equations below:

$$d^c_{ij} = \underline{\mathbf{u}}^c_i \cdot \underline{\mathbf{u}}^c_j \qquad \underline{\mathbf{C}}_{ij} = \underline{\mathbf{u}}^c_i \times \underline{\mathbf{u}}^c_j$$
$$d^t_{ij} = \underline{\mathbf{u}}^t_i \cdot \underline{\mathbf{u}}^t_j \qquad \underline{\mathbf{T}}_{ij} = \underline{\mathbf{u}}^t_i \times \underline{\mathbf{u}}^t_j$$

$$A_{ij} = \mathrm{sgn}(d^c_{ij} - d^t_{ij})$$
$$B_{ij} = [\mathrm{sgn}(\underline{\mathbf{C}}_{ij} \cdot \underline{\mathbf{T}}_{ij}) + 1] / 2$$

$$\underline{\mathbf{u}}_{ij} = (1 + B_{ij}(A_{ij} - 1))(\underline{\mathbf{u}}^c_i + \underline{\mathbf{u}}^c_j / \| \underline{\mathbf{u}}^c_i + \underline{\mathbf{u}}^c_j \|)$$
$$\underline{\mathbf{h}} = \Sigma \underline{\mathbf{u}}_{ij} \quad \text{where } i \neq j$$

[Koku, 2003]

Where $\underline{\mathbf{u}}^c_i$ is the unit vector pointing to a landmark i on the SES, while $\underline{\mathbf{u}}^t_i$ points to a landmark on the LES (c for current, t for target. A_{ij} represents the relative magnitudes of the angles between the pairs from the SES and LES". While "B_{ij} represents the ordering of the pairs". $\underline{\mathbf{u}}_{ij}$ is the resultant heading for each pair, and finally $\underline{\mathbf{h}}$ is the final heading. [Kawamura, 2002].

Also, it would be useful to refresh quickly what the discovered limitations of 2D ENav were:

1) Three Landmarks are required in order for robot to find goal
2) Convergence of robot to goal dependent on goal/landmark configuration (Convergence is not guaranteed)
3) Paths are often non-optimal and inefficient

At this point the progressive design and simulation of the 3D ENav system is discussed.

Design and Simulation of 3D ENav

For this system, and 3D ENav in general, the model of the egosphere returns to that of a 3D sphere. This means that the landmarks are located by two angles (Azimuth and Elevation), or else 3-Dimensional unit vectors.

The goal now is to expand ENav into a new 3-Dimensional version that makes use of this extra information. The first impulse is to simply apply the same vector math above to 3-Dimensional unit vectors. Specifically, from the Azimuth and Elevation angles, create 3-Dimensional unit vectors that represent the directions from the robot to the landmarks ($\underline{\mathbf{u}}^c_i$ for SES $\underline{\mathbf{u}}^t_i$ for LES).

This would seem to settle things quickly.

However this method outputs a heading that is 3-Dimensional as a result. Consider the two-landmark case. The graph below displays the unit vectors that point from the robot to the two landmarks on the SES (the blue lines), the same two on the LES (the red lines), and finally the computed heading (the black line).

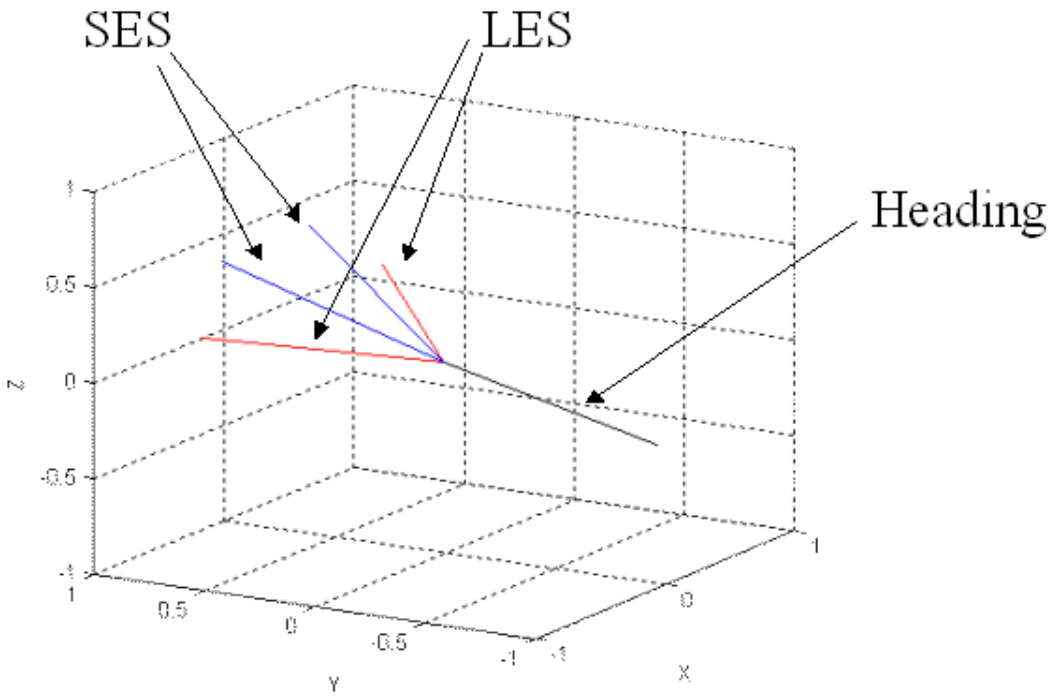

Figure 38. Heading Computed from 3D Inputs

Notice that the resultant heading points downward, in addition to away. Since the robot in consideration can only move along the surface of the floor this extra dimension is useless. If however we were using a robot that could move in three dimensions, the (UUV mentioned above), then there might be a purpose. However, in the current case, the z-component of the heading must be zeroed out.

The question is then; does the algorithm (ENav) perform any better in test cases in this new three-dimensional form than it did when we simply ignored the elevation angle (2D ENav)? From simulation, the answer is no.

To demonstrate this, a simulation program in MATLAB and C++ was written. [MATLAB] Essentially, MATLAB simulates a robot in an environment with landmarks and

a goal. The locations of all of these are defined in Cartesian space. Next it builds the SES based on the robot's location and the landmarks, and the LES based on the goal's location and the landmarks. Next it inputs this SES and LES into the ENav routine that outputs a direction. It then uses this direction to update the location of the robot. Then it acquires a new SES based on the robot's new location and reevaluates ENav, and so on until either a certain number of steps are performed (failure), or it comes within a pre-specified range of the goal (success).

While it does this, it outputs the locations of the goal, landmarks, and all the locations of the robot to a text file. These are then used by a C++ program called SIM, which was written for this research. SIM then creates a display for the simulation run using a free software package called CImg. SIM includes in this display circles of interest, specifically those that contain the landmarks and the goal.

Now, using SIM, we can test the success of this first try at 3D ENav. We can build a scenario where there is a robot, a goal region and two landmarks. This arrangement is shown below:

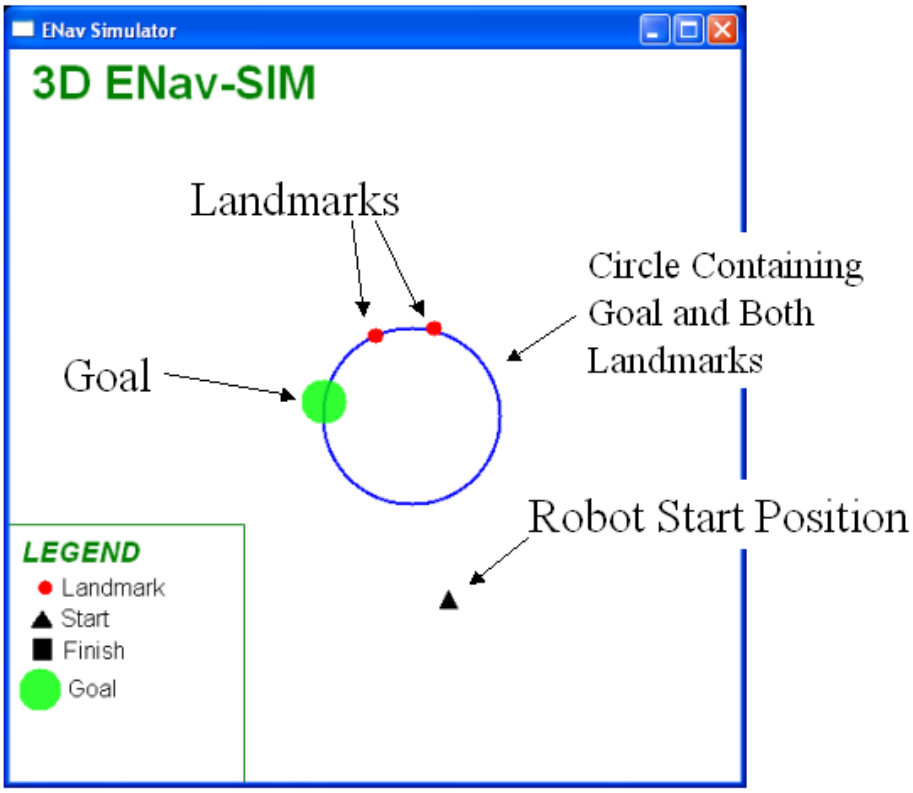

Figure 39. Simulator Screen

This is the display of the simulation, the C++ program SIM. The simulation represents a top-down view of a room. Displayed in this scenario are the robot's starting location, the goal location, and the two landmarks. Also shown is the circle that contains both landmarks and the goal, the reason for showing this circle is it sheds light on certain behaviors of the robot.

We can first run the simulation with the robot set to compute a path using 2D ENav for a quick demonstration of how this program works.

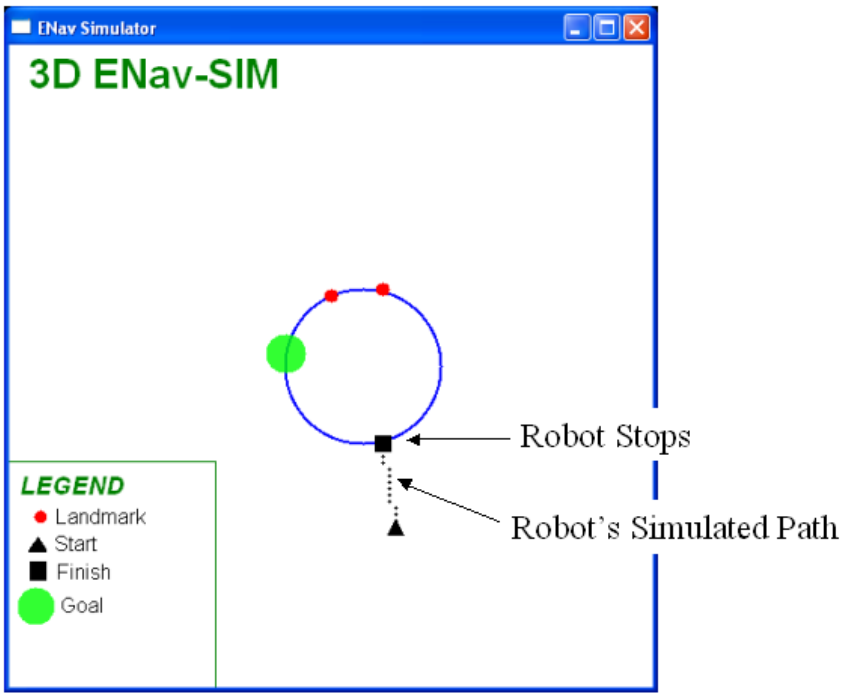

Figure 40. Simulation with Robot Running 2D ENav

In 2D ENav, everything is assumed to occur on a 2-Dimensional surface. The robot moves to make the SES resemble the LES of the goal region. However, in this 2-landmark case, once it reaches the circle which connects the goal and both landmarks, the SES and LES look the same although it is not at the goal and the simulation terminates without the robot reaching the goal.

At this point we can test the applicability of applying 3-Dimensions to Bugra's 2D Vector-based algorithm. We do this by assigning some Z value to the landmarks. Although this will not be reflected in the simulator, it will cause the elevation angles of both the robot's SES and the LES to be non-zero. It will also cause, as discussed before, the resultant heading to contain a Z component. This will be ignored. The simulation reveals how well this method performs.

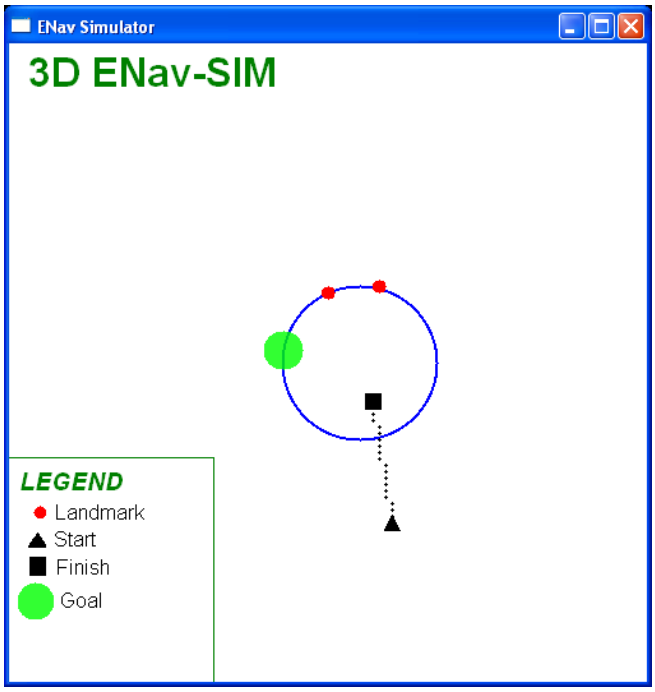

Figure 41. Applying 3-Dimensional Vectors to Original Algorithm

We see in this simulation that the robot doesn't seem to be heading any place in particular. It terminates at a position that is by all appearances random. It seems then that merely forcing an algorithm that produces a 3-dimensional heading to remain on a plane results in unsatisfactory behavior.

Instead of simply plugging in a 3-dimensional vectors into the old algorithm, it is probably better to design a new algorithm which is built around the 3-Dimensional SES.

An important realization is that the elevation angle can be compared against a known direction: up [From Dr. Wilkes]. This is a useful bit of information. If it is also taken into account that landmarks will always be above the robot, by at least some small margin, then it can be concluded that the larger an elevation angle is, the further away or higher the

landmark is. If it is assumed that actual landmark height is unchanging then the elevation angle and distance to landmark become directly related.

Knowing this we can construct a new experimental ENav that operates on elevation angles alone. Specifically it compares the elevation angles of the LES with those in the SES, landmark by landmark. In each case a unit vector is assigned either towards or away from the landmark's projection onto the floor depending on the result of the comparison. So if landmark one has a larger elevation angle in the target LES than it does in the current SES, a unit vector is assigned as pointing toward the projection of the landmark onto the base of the SES (so as not to assign three dimensional motion). After this is done for each landmark, all the assigned unit vectors are summed to derive a final heading.

To formalize a little bit, let Ev^c_i be the elevation angle for the current landmark i (SES), Ev^t_i be the elevation angle for the target landmark i (LES) and $\underline{\mathbf{u}}^c_i$ be a 2-D vector pointing to the projection of landmark i onto the base of the SES, then:

$$S_i = \mathrm{sgn}(Ev^t_i - Ev^c_i)$$

$$\underline{\mathbf{e}}_i = Si * \underline{\mathbf{u}}^c_i$$

$$\underline{\mathbf{h}} = \Sigma(\underline{\mathbf{e}}_i)$$

We can call this method "Elevation Egocentric Navigation". This simple algorithm produces rather good results in simulation:

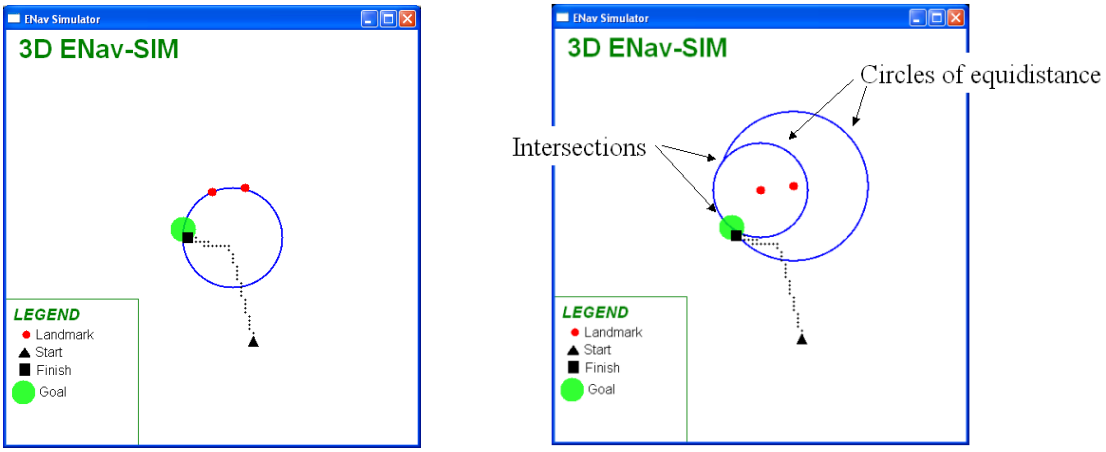

Figure 42. Elevation ENav

In this simulation run, using the above "Elevation ENav" algorithm the robot finds the goal. The path now however ignores the circle containing the goal and both landmarks. The path is more related to the circles that mark the locus of points that have distances equal to the distance from the landmark to the goal. These could be called the "Circles of Equidistance", which are shown on the right.

By displaying these "circles of equidistance", it becomes clearer why the robot takes the path that it does. It first can moves simply towards the landmarks, as they are both "attracting" the robot. However, when it crosses the boundary of the first circle, now one of the landmarks repels while the other attracts. The vector sums leads to a general motion toward the left, until it finally stops at the goal.

Noticing that the goal naturally occurs at one of the intersections between the two circles, it is realized that there is another intersection. This means that there exists another point in the scenario where the elevation angles for both landmarks match in SES and LES.

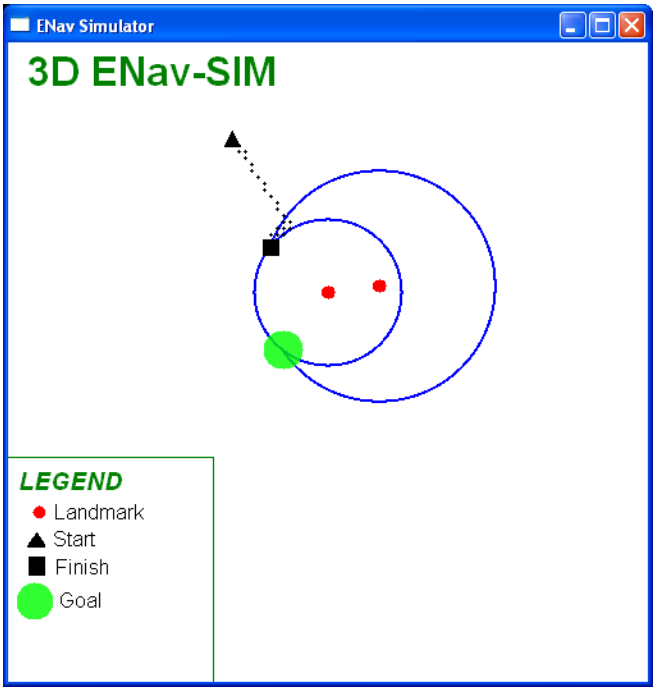

Figure 43. Simulating Elevation ENav

Simulating confirms that this point in the scenario, according to the elevation ENav algorithm appears identical to the goal region. However, this shows that this method, in the case of two landmarks converges either toward the true goal, or else a single false goal.

A solution to problem of the false goal could be now to combine this new method of navigation by comparing elevations, with the old method of 2D ENav. We choose the original 2D ENav over the version where we used 3D vectors in Koku's algorithm because we know that the original converges to the circle that contains the two landmarks and the goal. If we sum the directions given by 2D ENav, and this new elevation based approach, it should be that the robot could converge only on a single point: the goal.

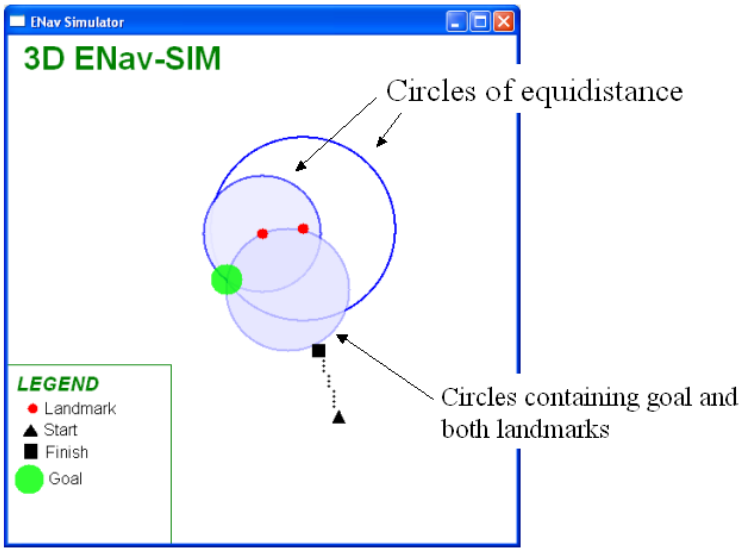

Figure 44. Simulating Combined 2D ENav and Elevation ENav

This method seems to have failed. However, the combined system of 2D ENav, with Elevation ENav is more incomplete that it is defective.

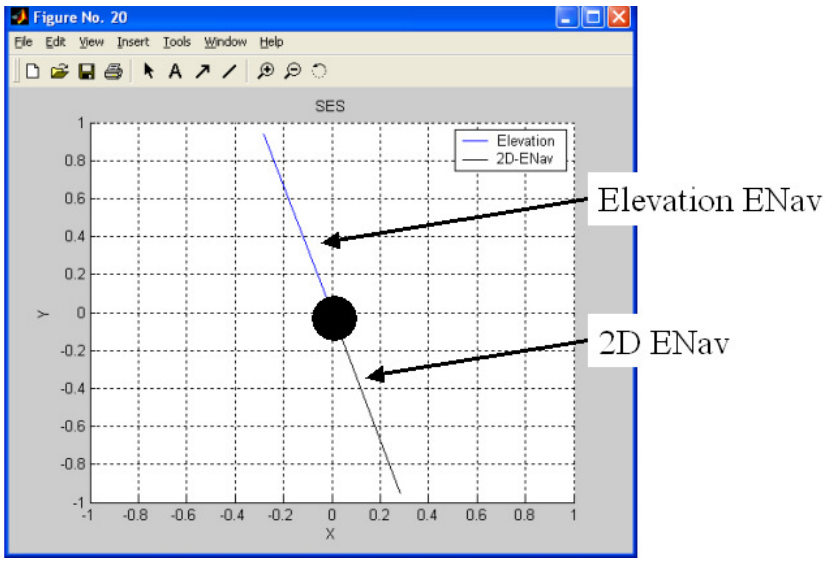

Figure 45. Computed Headings for Elevation ENav and 2D ENav

If we look at the above chart we the problem, at the point where the robot stops, the two directions are canceling each other out. We see that elevation algorithm, having crossed neither of the circles of equidistance, is pulling the robot towards both landmarks (which leads to pointing along the bisector), while the 2D ENav having reached the circle which contains both landmarks and the goal, is repelling the robot from moving any further towards the landmarks, which would move the robot away from the circle. The result is stagnation.

A good solution to this problem is to introduce error functions. An error function would describe how far each algorithmic calculation (2D ENav and Elevation based) is from equaling the values in the goal region. In the case of the elevation-based vector, this would be some measure of the difference between the elevation angle on the SES and the elevation angle on the LES. This could be implemented by a percent error function. Koku defines 2D ENav error in the case of two landmarks as:

$$E = |\alpha_c - \alpha_t|$$

Where α_c is the angle between the landmarks on the SES and α_t is the angle between the landmarks on the LES. [Koku, 2003] To be consistent, divide by the larger α term so that it too is a percent error function.

These error functions can now be used to scale the direction vectors given by each process. The result of this should be that of suppressing directions of low error, while amplifying directions associated with high error. In other words, give preference to the direction that is associated with the larger error.

If we implement these error functions we see the following result.

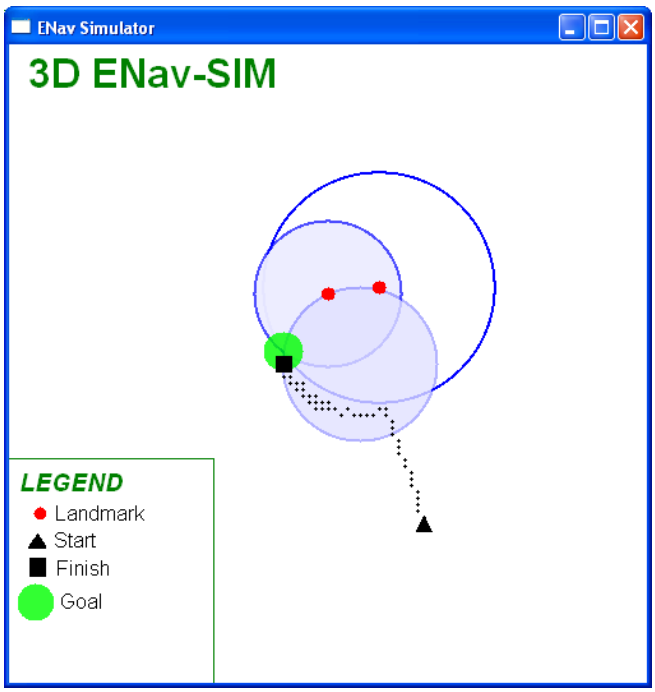

Figure 46. Combined 2D and Elevation ENav with Error Scaling

Now the robot succeeds in finding the goal. The path it takes reflects the delicate balance between the direction given by ENav (away from the landmarks, along the bisector), and by elevation (toward each landmark, weighted individually).

There is still another problem with this algorithm however, namely, that it still remains possible for the robot to fall into traps. This happens specifically when the robot is on the wrong side of the landmarks. In this case, 2D ENav will move the robots toward the landmarks to correct the backward ordering of landmarks, while Elevation ENav will repel as the robot gets to close to the landmarks.

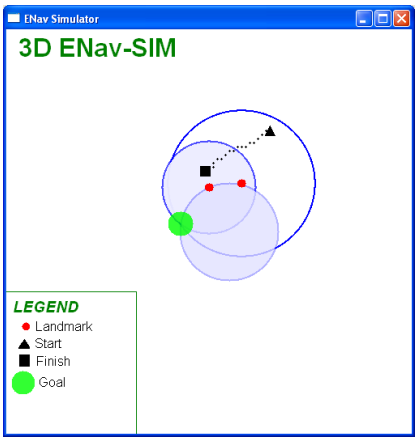

Figure 47. Stalling in Combined 2D and Elevation ENav

This is the case shown here. The robot becomes stuck between trying to move along the bisector towards the landmark (ENav) and moving away from the landmarks to the other intersection of equidistance circles (elevation). The solution to this is to say that because 2D ENav can detect that it is on the wrong side of a pair of landmarks, have the program inhibit the elevation algorithm until the robot is on the right side of the landmarks.

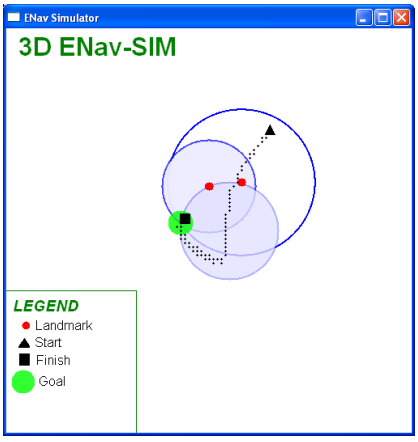

Figure 48. Combined 2D and Elevation ENav with Elevation Suppressed on Other Side of Landmarks

We see now that the robot relies on 2D ENav to position itself on the right side of the landmarks, and then proceed to the goal in a related way to as above.

At this point, the last modification to make is to change the algorithm from weighing the 2D ENav result against the average of the two elevation results into weighing all three results even. The reason for this change is that experience has shown this the superior technique.

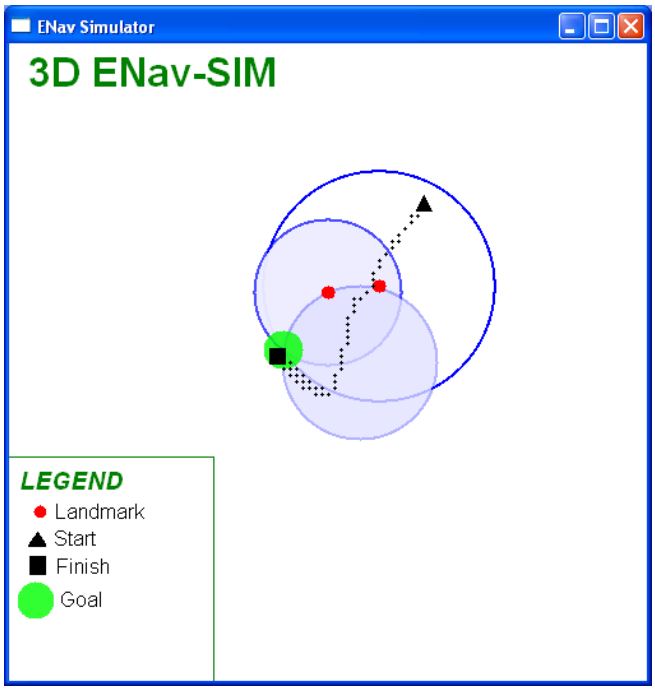

Figure 49. Combined 2D and Elevation ENav with Adjusted Scaling

Now, after testing, I believe that for the two-landmark case, this hybrid algorithm will always find the goal. The only degenerate case found is the case in which the landmarks and goal are collinear with the goal on one side, but exact colinearity should be rare (if not impossible) in the physical world, and if the landmarks are collinear on one side of the goal with the goal, then the robot will only be able to perceive one of the landmarks and the problem is moot.

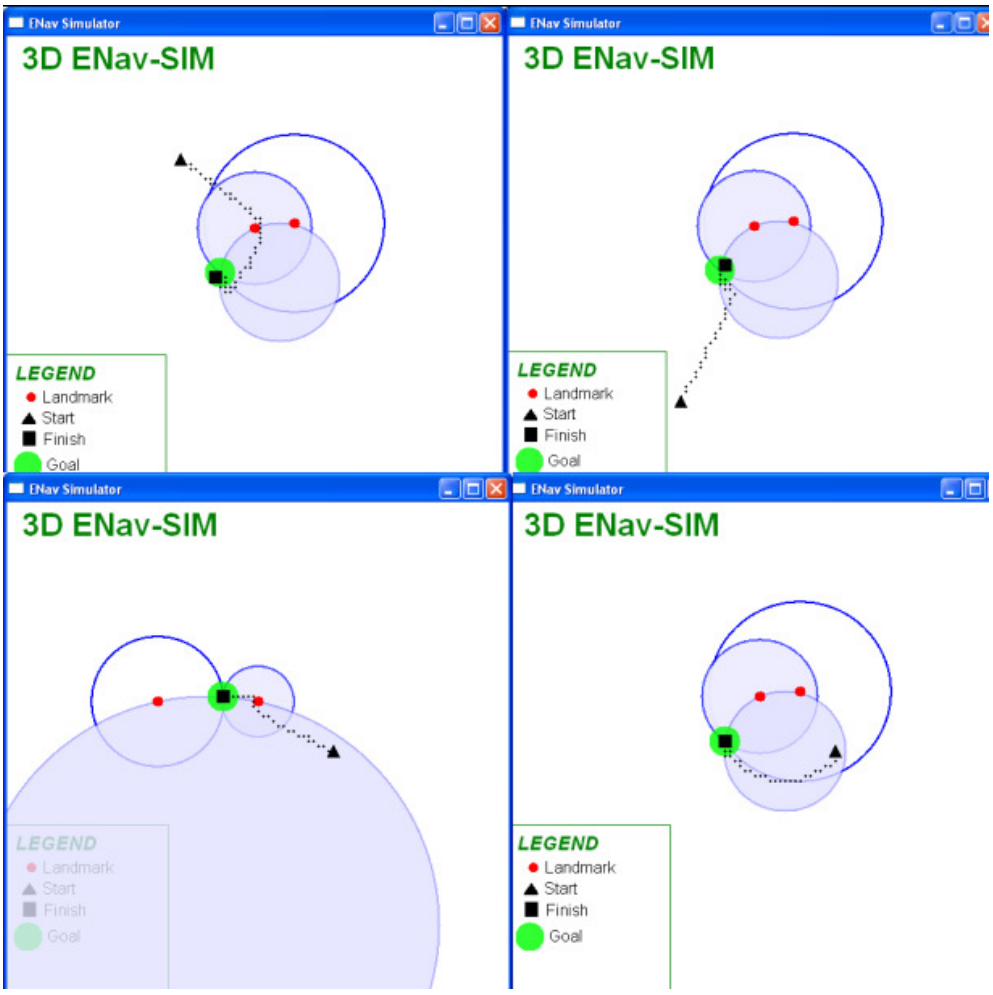

Figure 50. Test Runs of Combined 2D and Elevation ENav

Calculating error functions has the further benefit of providing the robot with an informed stopping point. Namely when some total error value goes below a predefined threshold, the robot is at the goal. The simplest total error function is the average of the separate error functions.

At this point, it is useful to formally define this new method of 3D ENav. Then this formal definition can be used to expand to the three and up landmark cases. Because this method is truly a hybrid of two earlier discussed methods, the definition could be best broken down into two sections.

The first portion of the description will calculate 2D ENav for each pair of landmarks. This is done according to the original algorithm. Remembering that the unit vector $\underline{\mathbf{u}}^c_i$ is the unit vector pointing to a landmark i on the SES, while $\underline{\mathbf{u}}^t_i$ points to a landmark on the LES, $\underline{\mathbf{u}}_{ij}$ is the resultant direction for each landmark pair, and $\underline{\mathbf{h}}$ is the final direction given by 2D ENav. The description is as follows:

$$d^c_{ij} = \underline{\mathbf{u}}^c_i \cdot \underline{\mathbf{u}}^c_j \quad \underline{\mathbf{C}}_{ij} = \underline{\mathbf{u}}^c_i \times \underline{\mathbf{u}}^c_j$$
$$d^t_{ij} = \underline{\mathbf{u}}^t_i \cdot \underline{\mathbf{u}}^t_j \quad \underline{\mathbf{T}}_{ij} = \underline{\mathbf{u}}^t_i \times \underline{\mathbf{u}}^t_j$$

$$A_{ij} = \mathrm{sgn}(d^c_{ij} - d^t_{ij})$$
$$B_{ij} = [\mathrm{sgn}(\underline{\mathbf{C}}_{ij} \cdot \underline{\mathbf{T}}_{ij}) + 1] / 2$$

$$\underline{\mathbf{u}}_{ij} = (1 + B_{ij}(A_{ij} - 1))(\underline{\mathbf{u}}^c_i + \underline{\mathbf{u}}^c_j / \| \underline{\mathbf{u}}^c_i + \underline{\mathbf{u}}^c_j \|)$$
$$\underline{\mathbf{h}} = \Sigma \underline{\mathbf{u}}_{ij} \quad \text{where } i \neq j$$

Next we calculate the headings from Elevation ENav (where Ev^t_i is the elevation for landmark i on the LES and Ev^c_i is the elevation angle of landmark i on the SES.)

$$S_i = \mathrm{sgn}(Ev^t_i - Ev^c_i)$$

$$\underline{\mathbf{v}}_i = Si * \underline{\mathbf{u}}^c_i$$

Where $\underline{\mathbf{v}}_i$ is the direction given by the Elevation ENav for landmark i. Now calculation of error functions, starting with ENav

$$e_{ij} = abs(\alpha^t_{ij} - \alpha^c_{ij})/max(\alpha^t_{ij}, \alpha^c_{ij})$$

$$E = avg(e_{ij}) \text{ where } i \neq j$$

Where α^t_{ij} is the angle between landmarks i and j on the LES and α^c_{ij} is the angle between i and j on the SES. For Elevation ENav there is:

$$ev\_er_i = (Ev^t_i - Ev^c_i)/max(Ev^t_i, Ev^c_i)$$

Now we can compute the total heading.

$$\underline{\mathbf{H}} = \underline{\mathbf{h}} * E + (\Pi B_{ij}) * (\Sigma (\underline{\mathbf{v}}_i * ev\_er_i))$$

In this formula, we are taking advantage of the face that if B_{ij} is zero for any case the robot is on the wrong side of landmarks and therefore repress the elevation directions.

We can call this method 3D ENav. Temporarily, the above formulation will define 3D ENav, but later work reveals modifications necessary to the algorithm. Now that the 3D ENav method is formalized, we can analyze its effectiveness when we expand to cases of 3 and 4 landmarks.

Analyzing the 3 landmark case

By adding another landmark to the scenario significant improvements are made. Specifically, because the 3D ENav system has more inputs, the 2D ENav component is now the average of 3 directions, while there is an extra direction in the elevation algorithm. This often leads to "better" directions in terms of optimality and directness. The following examples demonstrate. First, we consider a two-landmark scenario as follows:

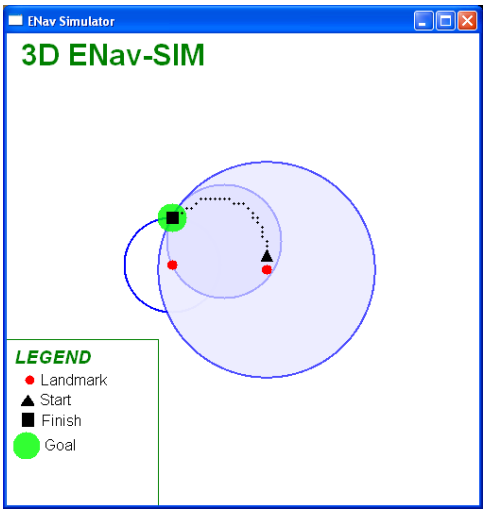

Figure 51. 3D ENav Simulations with 2 Landmarks

The path was computed using the algorithm described above. Notice that the path taken by the robot is not direct. Now if we introduce a new landmark to the scene, and apply the same algorithm, we get the following results:

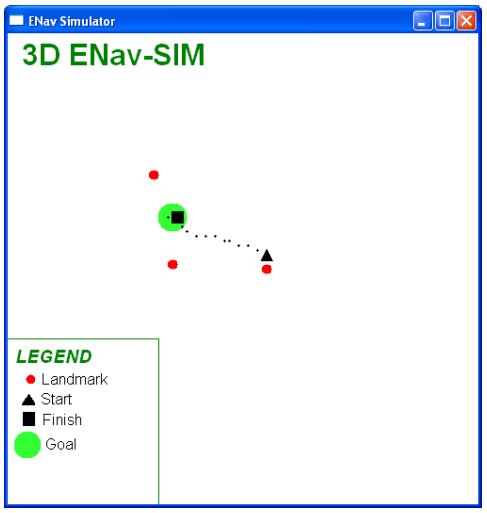

Figure 52. 3D ENav Simulation with 3 Landmarks

Because there would now be three circles of equidistance, and three circles containing two landmarks and the goal, they have stopped being shown. The relationship between the robots, goal and any one circle would not demonstrate much anyway. Notice that the robot progresses much more directly to the goal on account of the increased amounts of inputs. It would seem then that by adding more landmarks we have solved the problem of inadmissible paths.

However, there is a problem with simply adding more landmarks to the above-discussed algorithm. Remembering that the final 3D ENav algorithm is a hybrid of 2D ENav and an elevation method, it is worth considering what happens when the goal is in the so called Region III, outside the triangle of the landmarks. The following demonstrates:

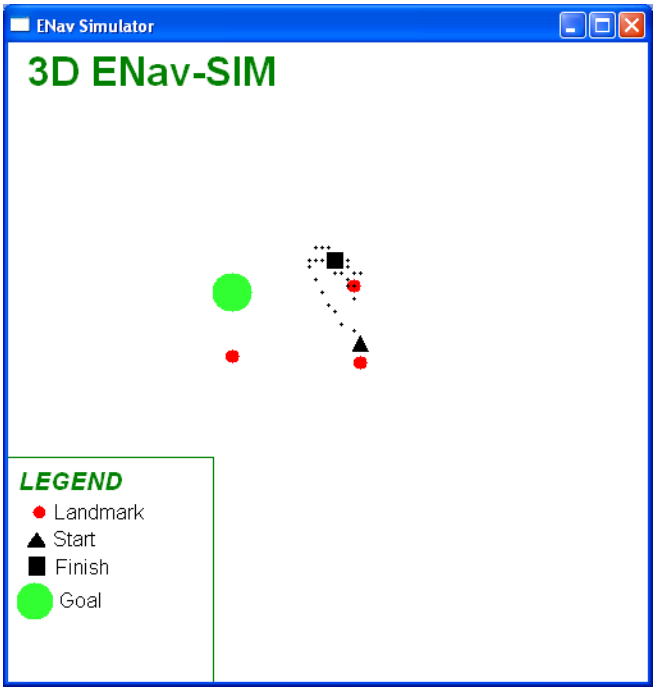

Figure 53. 3D ENav Simulation with Goal in Region III

The robot fails to converge in this case. This can be understood because the 2D ENav component diverges in this case, and while the elevation component converges, the 2 compete and lead to a stalemate.

The problem continues into the four-landmark case:

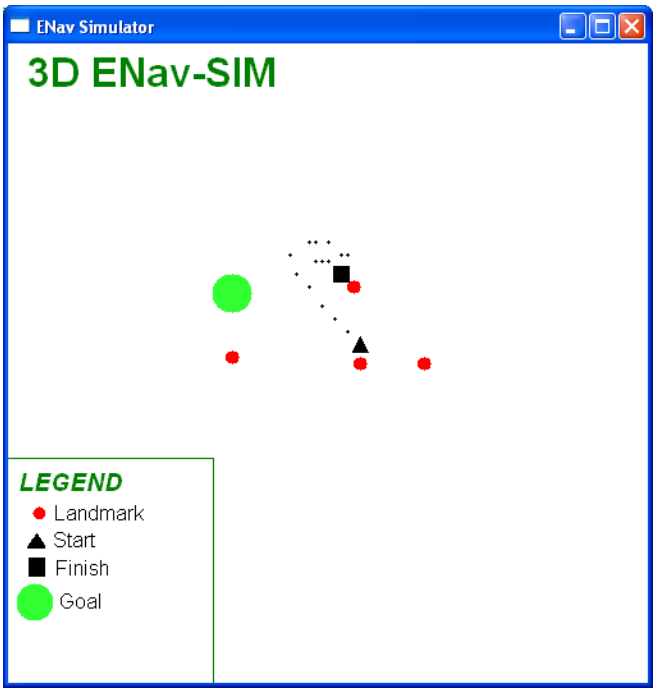

Figure 54. 3D ENav with 4 Landmarks

It becomes apparent that adding more and more landmarks cannot help this method. If the goal is in region 3 of the landmarks, the 2D ENav component will be supplying divergent headings.

The solution to this problem is to realize that the two-landmark case was convergent. If instead of simply calculating the algorithm for all the landmarks at once, if every pair of landmarks was computed using the two-landmark method, and then the results of each pair wise calculation averaged, the average should also be convergent.

To simulate this solution, the simulator was moved to C++, and remodeled the landmarks to match what would actually be used on the robot. The above solution was implemented, where the landmarks are computed pair-wise.

So then the final solution will be to implement the algorithm outlined above on each pair of landmarks, and then average the output of each pair for a final heading.

This new algorithm will now define 3D ENav.

If we set up the first 3-landmark scenario from above (the two landmark case is identical), the following results are obtained:

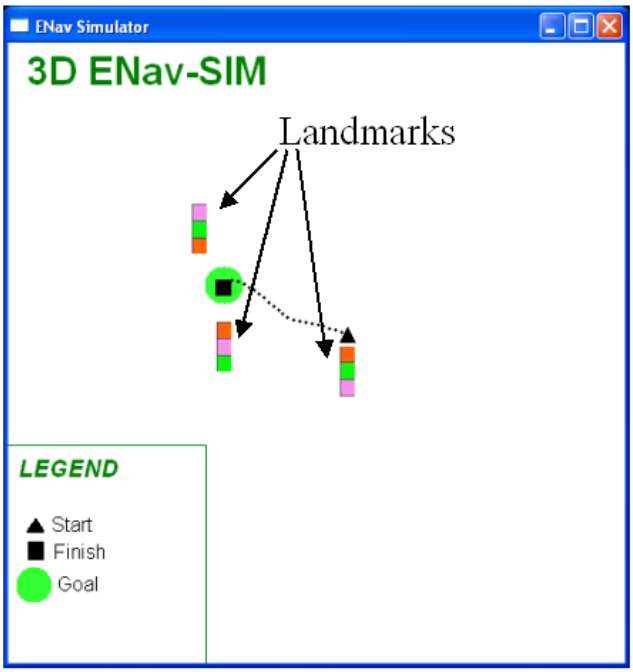

Figure 55. Retrying 3 Landmark Case

We see then that for the new algorithm, the robot also converges if the goal is in the triangle. Now, examining the case of the landmark in region 3 shows:

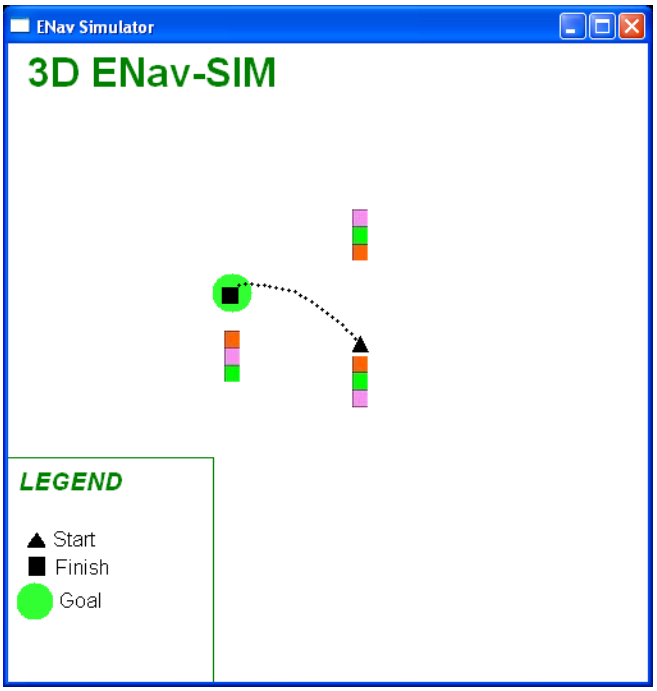

Figure 56. Simulating with Goal in Region III

Thus this new method demonstrates its ability to converge in the three-landmark case, and in practice, it converges for all realistic scenarios. The exception is again the case where all the landmarks are co-linear with the goal, and this is acceptable, as the visual system cannot perceive the landmarks in the set up anyway.

This method has the advantage in that more landmarks introduced into the scenario tend to increase the directness of path selected by the robot. In the above scenario, the path is fairly direct, however, introducing more landmarks shows:

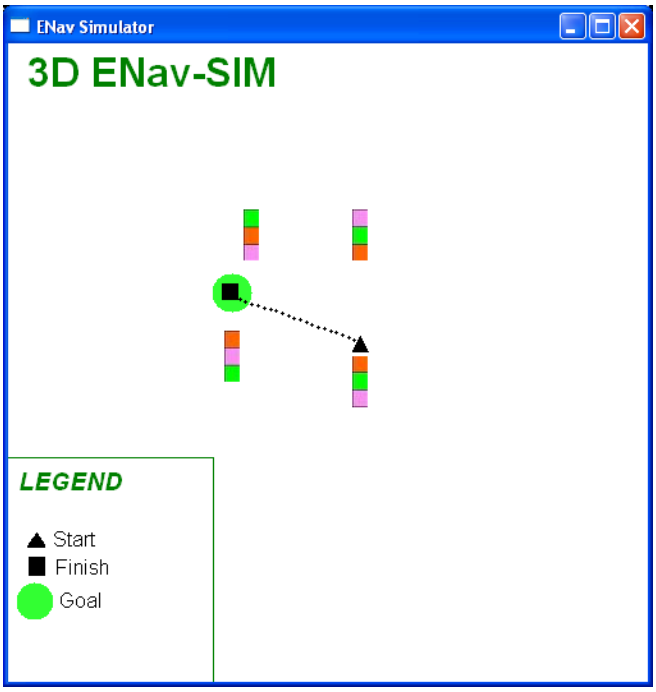

Figure 57. Simulating 4-Landmark Case

The fourth landmark makes the path very direct.

Reviewing the three main flaws in 2D ENav that 3D ENav tried to alleviate, we can mark the progress thus far:

The first problem described was that 3 landmarks were the minimum number of landmarks required in order for the robot to converge. It is shown that 3D ENav reduces this number to 2.

The second problem was that the robot was not guaranteed convergence even in the 3 landmarks and up case if the landmark was outside the region triangulated by 3 landmarks. It is shown that for the 3-landmarks and up case, convergence is essentially guaranteed for all realistic configurations.

The final problem dealt with paths. In the three, four and up landmark case, the paths are generally pretty, although not totally optimal. In the two-landmark case however, the paths can be very indirect. Consider the demonstration below:

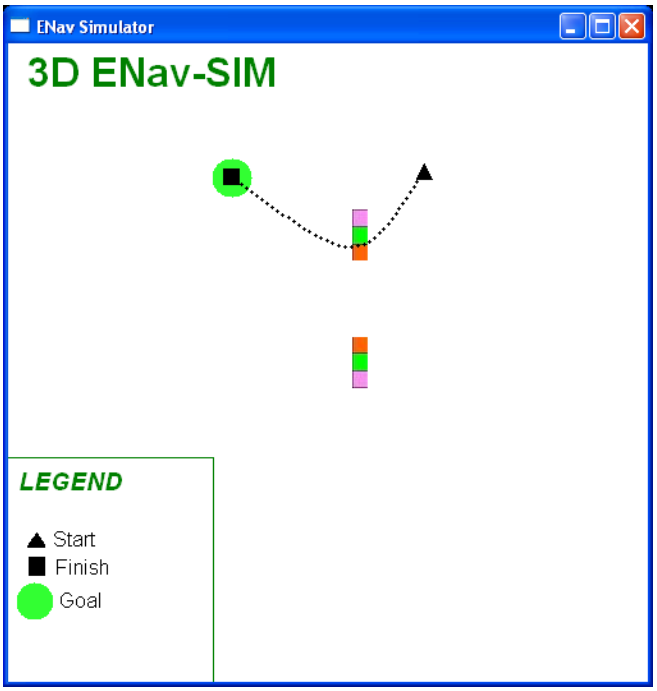

Figure 58. 3D ENav with 2 Landmarks

In this case, although the robot navigates to the goal, it does so in a less than optimal way. In fact, paths this bad are generally limited to the 2-landmark case. This can be seen by adding another landmark.

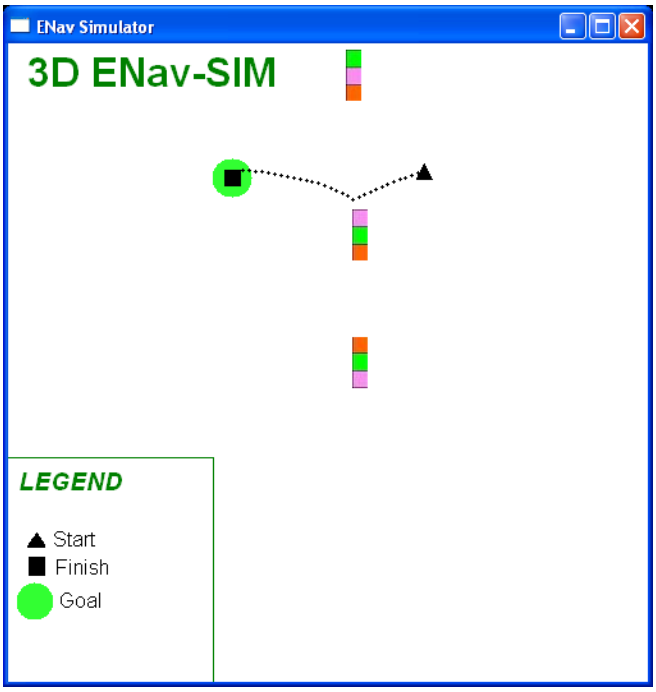

Figure 59. Simulation with Added 3rd Landmark

However, even this path is less then perfect. To combat this final problem, Dr. Wilkes and I built into ENav, the capability to guess at a more intelligent path than the one given by direct 3D ENav. This is described in the next section.

3D ENav +

The final problem afflicting 2D ENav for 3D ENav to solve is that of un-optimal paths. Although 3D ENav certainly improves the directness of paths in most cases, it too can yield un-optimal paths. The solution presented here is to allow the robot to simulate 3D ENav internally through to convergence, and then move towards the point predicted internally to be the goal.

To derive this method, begin by assuming that the robot is given an SES and an LES. For this method, start by assuming that the robot is also given the heights of the landmarks (we can relax this assumption later on). Now, if the robot knows the SES and the heights of the landmarks then it knows the distance and direction of all the landmarks, relative to itself. This is because the elevation angle and the height can be combined to find the distance using: distance = height / tan(elevation)

Now, using this information, the robot can build an egocentric map of its surroundings. It puts itself at the center, and the landmarks are placed based on their distances and azimuth angles.

Next the robot computes the direction given by 3D ENav using the SES and the LES given at the beginning. After doing this, instead of actually moving, it moves the virtual robot on the map according to the result. Now it uses the map to build a new SES based on the virtual robot's location. It repeats this process until the virtual robot reaches a point where its error between SES and LES is below some value, or else a pre-set number of iterations are passed. If the virtual robot does converge on a goal, the vector that points from the virtual robots original location, to the virtual robot's final location is returned as the direction to move in. If the virtual robot does not converge, then the result of "regular" 3D ENav based on the original SES and LES are returned.

This method works very well in simulation. However, the condition that the heights of the landmarks are known in advance is too constricting. It would be better if the robot could estimate those heights itself, than the algorithm would work given the exact same information given "regular" 3D ENav.

Dr. Wilkes and I devised a method for this height estimation. Essentially, the robot stores its current SES, moves forward a known distance, and then grabs a new SES and through comparison determines the height of all the landmarks in both SES's.

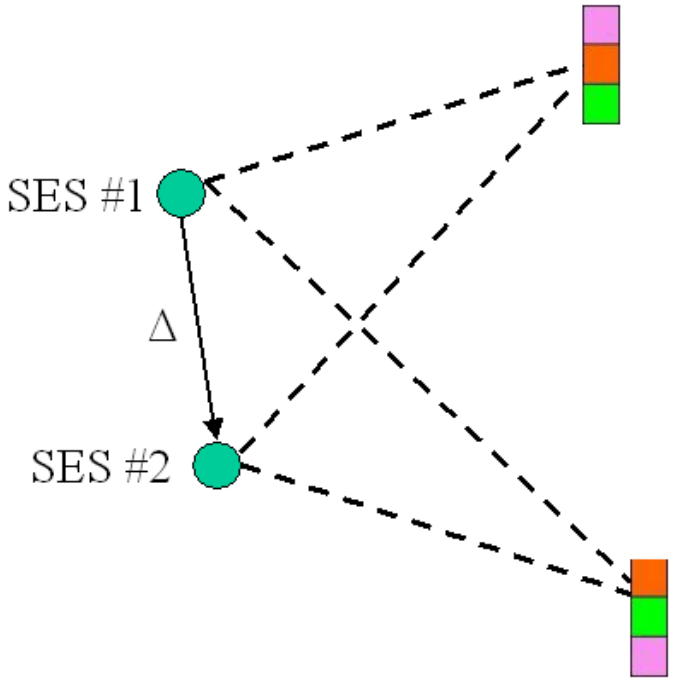

Figure 60. Estimating Heights of Landmarks Through SES Calculations

Here we see the movement of height estimation. The robot grabs SES #1 (which contains an azimuth and elevation angle for each landmark), moves a known distance Δ, and then grabs SES #2. Now if we consider just one landmark and define θ_i as the elevation angle at SES i, and ϕ_i as the azimuth angle form SES i, d_i as the distance to the landmark from SES i and h as the height of the landmark the situation can be drawn in 3D as follows:

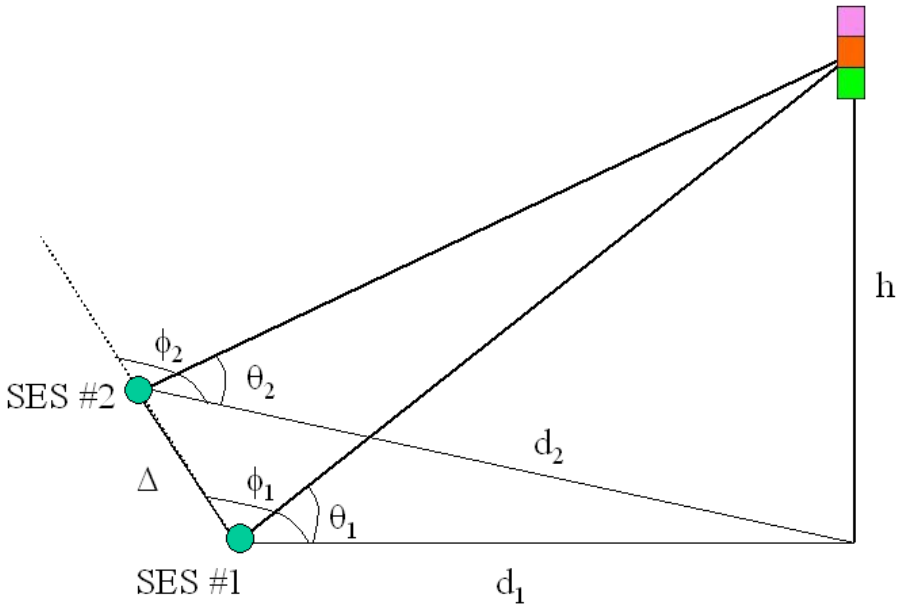

Figure 61. 3-Dimesional Visualization of Height Estimation

Also, if we observe this scenario from directly above we can see that:

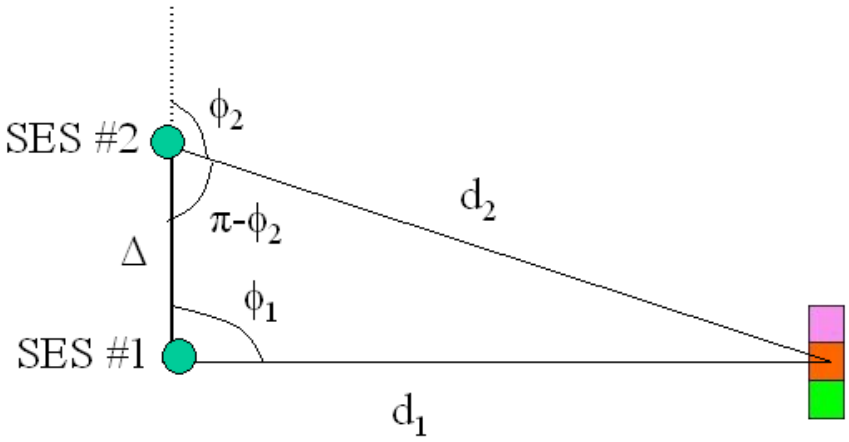

Figure 62. 2 Dimensional Visualization of Height Estimation

Now we can use the law of sines:

$$\frac{a}{\sin(A)} = \frac{b}{\sin(B)}$$

where a and b are sides of a triangle, and A is the angle opposite side a. Using this we find that:

$$\frac{d2}{\sin(\phi_1)} = \frac{d1}{\sin(\pi - \phi_2)} = \frac{\Delta}{\sin(\phi_2 - \phi)_1}$$

this gives that

$$d1 = \frac{\Delta \sin(\pi - \phi_2)}{\sin(\phi_2 - \phi_1)}$$

and that

$$d2 = \frac{\Delta \sin(\phi_1)}{\sin(\phi_2 - \phi_1)}$$

now h can be derived using the 3-dimenstional drawing to be

$$h = d_1 \tan(\theta_1)$$

and also

$$h = d_2 \tan(\theta_2)$$

we can average these two values to arrive at a guess at the height of the landmark.

There are a couple of extra provisions to this method. The first is that both azimuth angles are made positive before hand, if they were not already. Also, because the robot is

moving forward, and only a short distance, it is most likely that the landmark will not pass one side to the other. However, should this occur, the method is disqualified.

Now with height estimation, the method is complete.

Now we can simulate the performance of a robot with this expanded 3D ENav, which is called 3D ENav+.

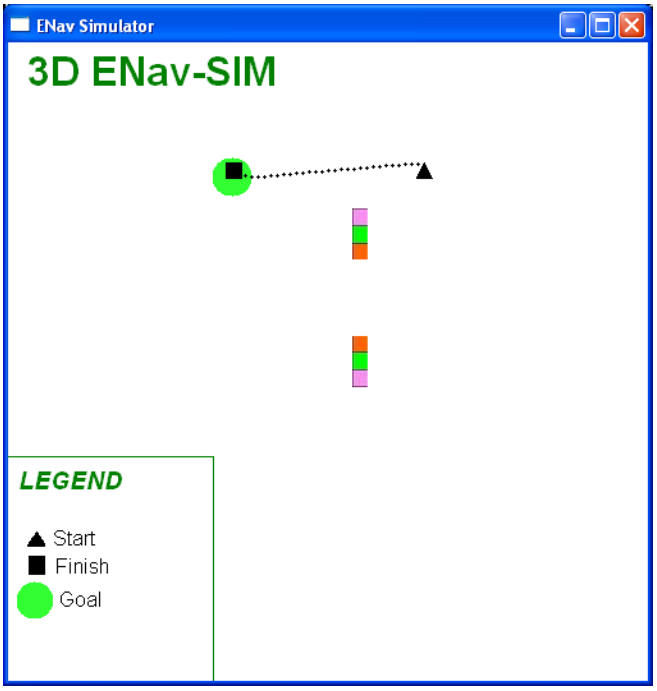

Figure 63. Simulation of 3D ENav+

At each point along the path, the robot simulates its convergence to the goal, and then moves toward the predicted goal region. This causes the robot to move directly toward the goal, even in the 2-landmark case. It is useful to point out that although the robot is using an

egocentric internal map, the robot is given the exact data used to calculate the original 3D ENav algorithm, and it just spends more time processing to arrive at a better path.

In estimating the heights there are two methods available. The first is to start navigation by estimating the heights, and then using the estimates for the rest of the run. The other method would be to continually estimate the heights across the run so as to constantly improve accuracy.

The first method is used in simulation. This is for two reasons. The first is that it is easier to do because the robot in the simulation is always facing up (this doesn't change the ENav calculations for testing purposes and is easier to code). So it makes sense to take an arbitrary step upward at the start in order to estimate the heights, and then not recalculate for simplicity. The other reason to do it this way in simulation is because the estimation will always be perfect; nothing is gained by constant re-estimation.

On the physical robot however, the second method is better. This is because the physical robot can move forward in any direction, and it can gain better estimates through repeated estimation.

Now with 3D ENav+, the robot needs only two landmarks, to converge on the goal, and to do so directly. A discussion of the hardware and software implementation of the navigation system follows.

Hardware

The hardware that encompasses the navigational section of the robot is one computer. Specifically, it is the main "off-robot" computer. This is the same computer that holds the local SES and LES memory structures. It is also the same computer that instructs the visual

system to take a scan, and the computer where the results of that scan are returned. This computer will be in general the main controller, but that will be discussed in more depth later.

Software

The navigational algorithm is contained within the C++ class ENav_C. This class is structured as follows:

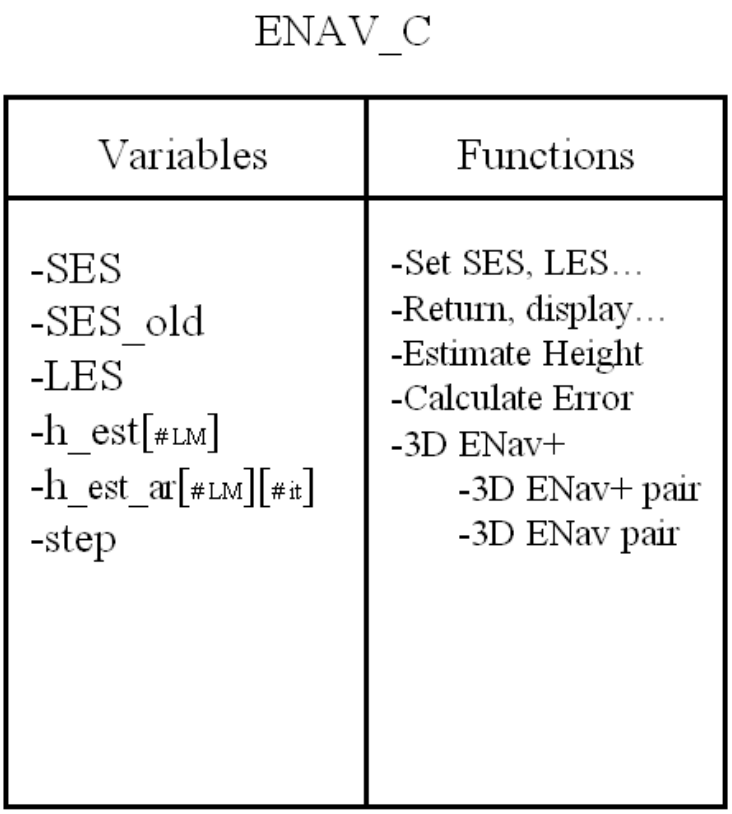

Figure 64. Structure of ENAV_C

The basic use of this class is that the user (or calling program) can store into the class an SES and an LES and then call 3D ENav+ which will return the direction. The user can also store in an SES_old, move forward a distance, store an SES, and then calculate the height through the Estimate Height function (which the user must pass delta, or distance traveled.) These height values are stored in the h_est_ar (height estimate array), for each landmark, for each iteration. The final h_est is recalculated every time the heights are estimated by averaging the h_est_ar for each landmark. The class can return the error between the SES and the LES. Finally there are return and display functions for many of the internal members.

The function 3D ENav+ uses sub-functions in order to operate. Specifically, it loops through all the pairs of landmark that exist in both the SES and the LES. For each pair it either calls 3D ENav+ pair or 3D ENav pair. Either function calculates a heading for the two-landmark case and returns a direction, which the 3D ENav function averages with all the other returned directions to derive a final direction to return to the user. The choice of sub-function depends on three conditions. First, the user can decide whether or not to use 3D ENav+ or just 3D ENav. Secondly, if the user opts to use 3D ENav+ but there is not a height estimate for one of the landmarks in the pair, then the 3D ENav function is called. Finally, if 3D ENav+ fails to converge, than 3D ENav is called after 3D ENav+.

The 3D ENav pair function (no +) operates given the angular indexes of two landmarks, for both current and goal states (taken from SES and LES). It then uses the algorithm for 3D ENav defined above to return a direction to 3D ENav.

The 3D ENav+ pair function is more complicated. It is also passed the angular indexes for two landmarks for current and goal. Additionally, it uses the height estimates stored in h_est for each landmark.

The 3D ENav+ function works to implement the simulated robot method discussed above. To do this, it uses the following steps:

1) Copy current SES is holding location
2) Build egocentric map using SES and height estimates in Cartesian space
3) Build new SES based on Cartesian map (will be same for first iteration)
4) Call error function, if below pre-set threshold restore original SES and return difference between current and stored SES
5) Call 3D ENav pair
6) Move simulated robot according to result of 3D ENav pair
7) While number of iterations less then pre-set maximum, go to 3

It is shown that the 3D ENav+ really virtualizes the member variables of the class in order to simulate the robot's movement toward the goal. To demonstrate 3D ENav+ consider the following example:

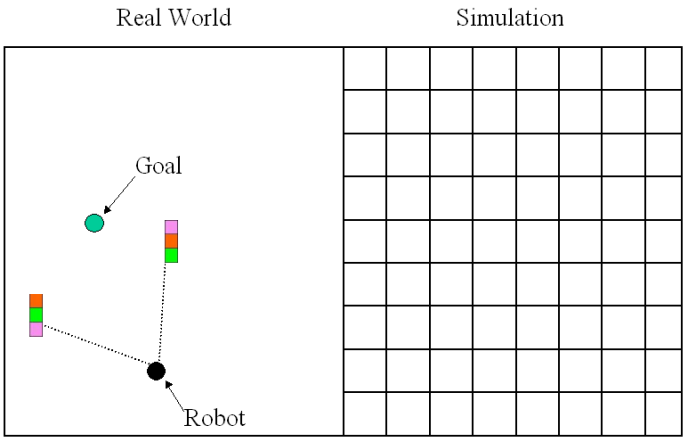

Figure 65. A Scene with 2 Landmarks

Here we see a scene with two landmarks, a robot on a goal. The robot can grab the SES from the scene. Assuming that the robot already has height estimates of the two landmarks, it can then build a simulated environment using geometry.

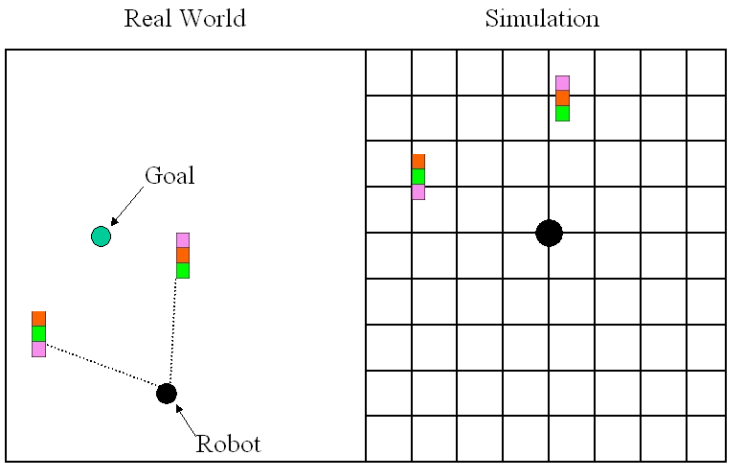

Figure 66. Build Simulated Scene

The robot can now grab SES's from the simulated environment, and then simulate movement in the simulated environment.

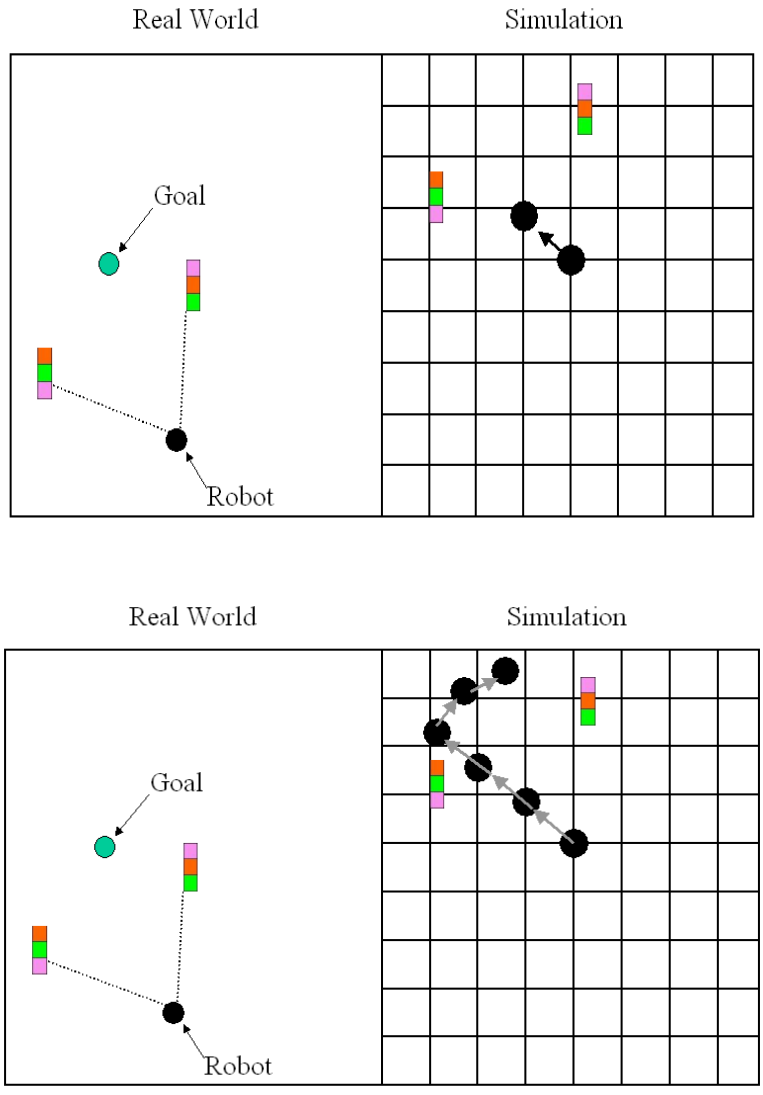

Figure 67. Moving in Simulated Environment

This function then returns the direction between the initial and final SES.

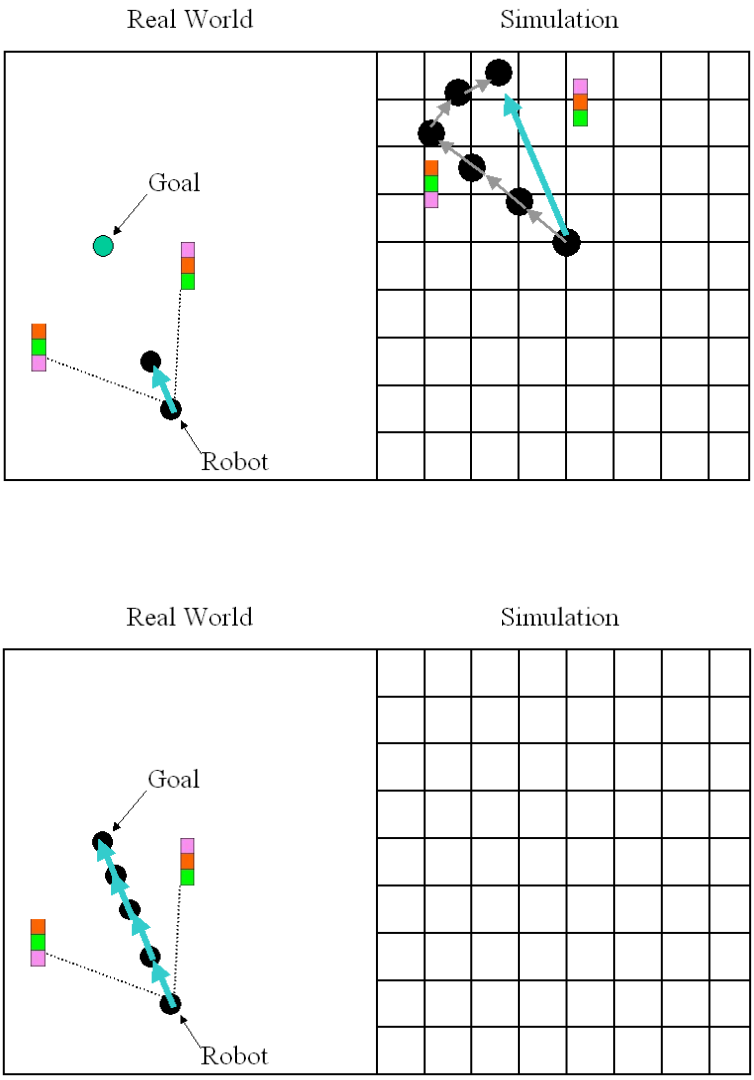

Figure 68. Returned Direction and Motion

The physical robot then moves in this direction. This action repeats until the error between the real SES and LES goes below the threshold value.

Notice that this results in a much straighter path for the robot to the goal. At this point we can defer full discussion of navigational implementation until we have compiled the complete system.

Motor Control

Hardware

Motor control is the sub-system for the movement of the robot. For this system, the pioneer robot (the platform being used) is controlled through a laptop computer on the robot. The laptop is running player [http://playerstage.sourceforge.net/] and can receive movement commands from another computer. It can also read sensory data from the robots sensors. This leads to the following break down:

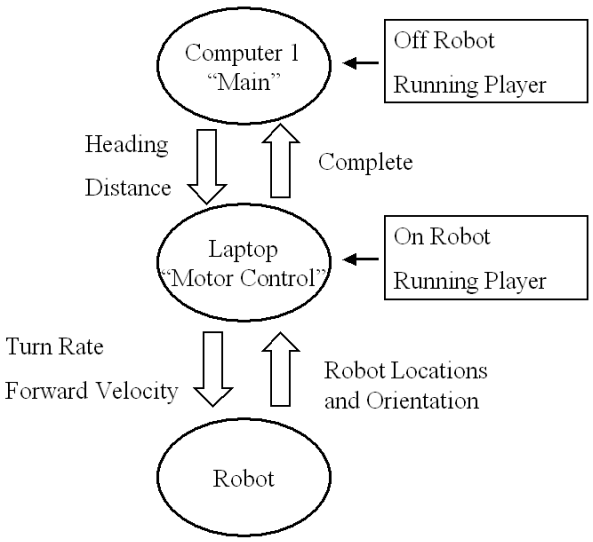

Figure 69. Motor Control Setup

It is shown that the main computer deals in simpler numbers, a direction for the robot to travel in, and the distance to travel. This is passed on to the Laptop through player, which converts this into commands for the Robot in terms of turn rates and velocities. The values are then carried out by the motors through Player.

Software

Player and the Pioneer essentially mainly implement the software for motor control. Very simple commands are used to control the robot based on the results of 3D ENav+. This is done through the hardware shown above.

Controller

Now we put all the subsystems together: the visual subsystem, the memory subsystem, the navigational subsystem and the motor control. It makes sense to introduce a new subsystem called the Controller to handle all the individual subsystems. We can look at the total layout then in terms of systems first.

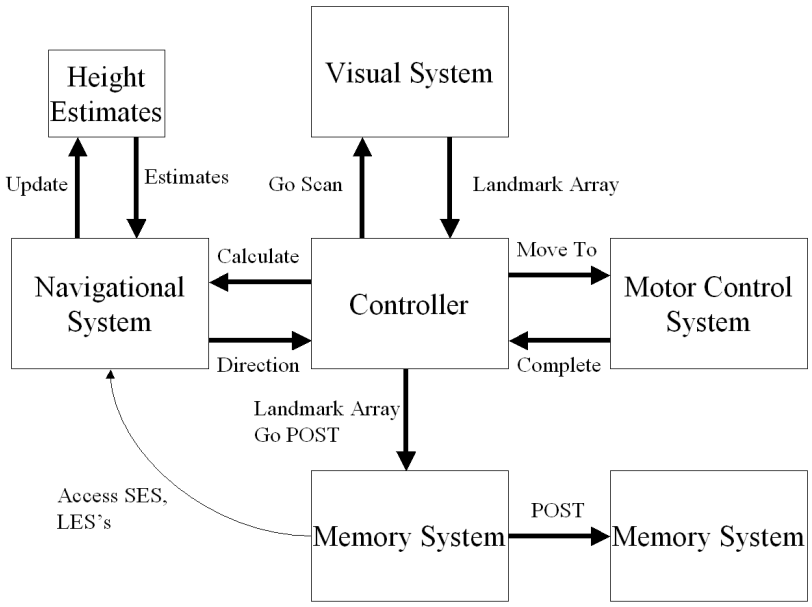

Figure 70. Total System Layout

Here is shown the overall layout of all the subsystems combined. Although this diagram isn't absolutely accurate, and doesn't show in full detail all the interconnections, it gives a good representation of the total system.

The controller is at the center of the subsystems. This block handles all the interconnections and ordering of operations. Its main function is to request each sub block perform one of its routines at the appropriate time. For instance, it can request a landmark scan, or a navigational calculation or a motion. The controller is designed to act in an abstract dimension, and not get involved in the details; these duties are intentionally delegated to the respective sub-systems. So for instance it can request a 3D ENav+ calculation and the navigational subsystem handles all the details of comparison, fetching height estimates, checking validities, etc.

Although there are a large amount of possible sequences to be implemented, there is one essential sequence. Specifically, the sequences that fully implements 3D ENav+ is the ones most concerning to us. A simplified sequence could be used to move without height estimation (3D ENav (no +)) by skipping height related steps.

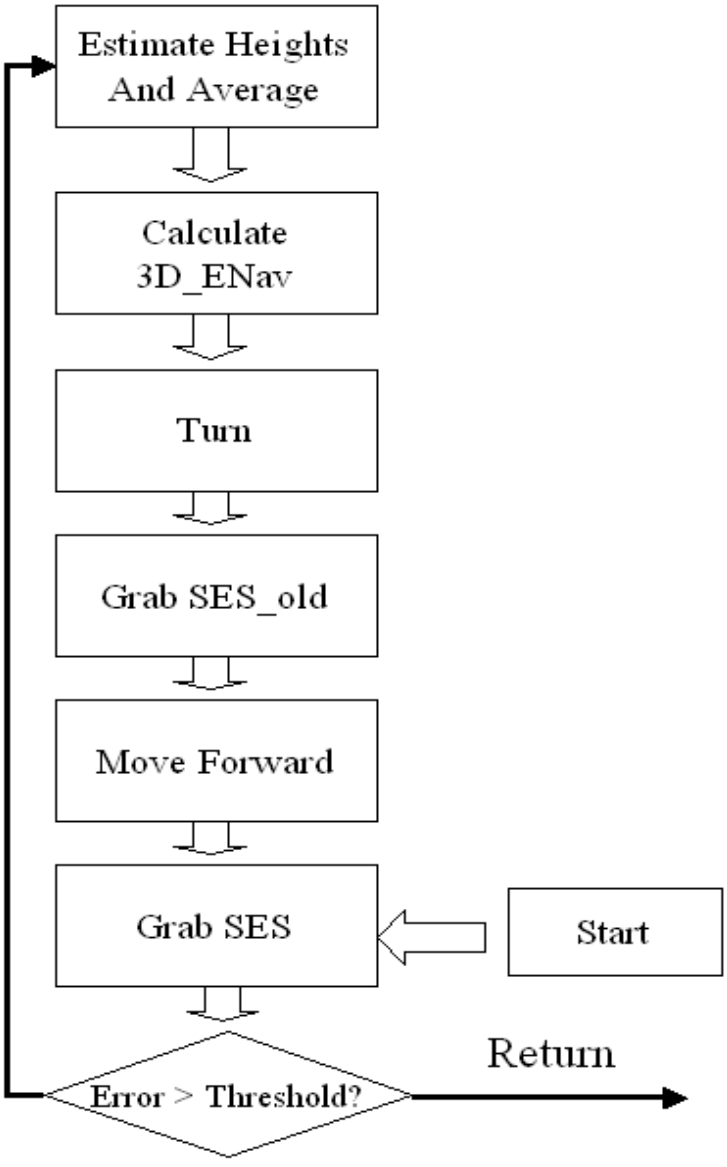

Figure 71. 3D ENav+ Sequence

Assume that the Robot has already loaded into the Memory subsystem LES's, and furthermore has selected which LES it intends to navigate to. Then the first step for the robot is to grab the SES, in other words perceive its environment. Using the error functions outlined in the Navigational section, it defines an error to decide whether it is close enough to the goal. If it is not, its next step is to estimate the heights of the perceived landmarks. Because this is the first iteration, this can't be done. Then the robot calculates 3D ENav+ given its most recent SES, and the LES. Because there are no height estimates, the navigational algorithm selects 3D ENav to compute a heading. Next the robot turns in the direction of the heading. Now it grabs an SES and stores it in SES_old, this will be needed to estimate heights in the next iteration. The robot then moves forward a pre-defined amount. It then grabs a new SES, and reevaluates whether it is close enough to the goal. If it is not, it uses the SES and SES_old, as well as the known distance of forward travel, to estimate the heights. Now it can use those height estimates to compute 3D ENav+ and the cycle continues until the error crosses some threshold.

To demonstrate this, the following is an example of the full sequence.

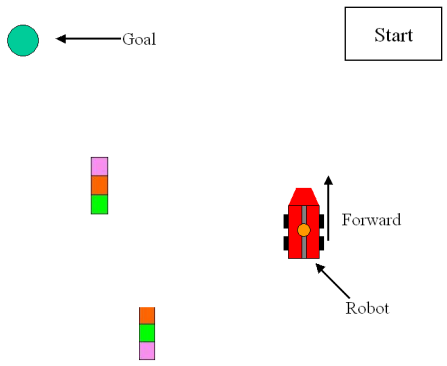

Figure 72. A Scene with 2 Landmarks

Here in this scene is a robot with two landmarks. The robot currently has its goal LES loaded, but has no height estimates. According to the sequence, the first step is to grab the SES:

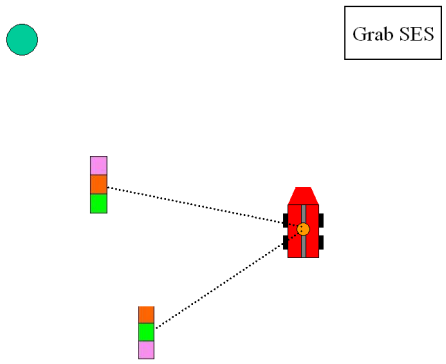

Figure 73. Grabbing the SES

The controller tells the Visual System to grab the scene, and the Visual System returns the Landmark Array. This is passed into the Memory System where it is converted and stored as an SES. Next using the error functions in the Navigational System, the robot determines if it is close enough to the goal, which it is not. Since there is nothing stored in SES_old, the robot skips the height estimation step and calculates its direction based on 3D ENav (not +). This returns a direction based on the SES and LES alone.

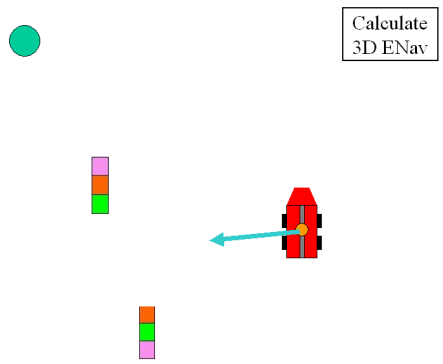

Figure 74. Returned 3D ENav (not +) Heading

The robot now turns itself toward this direction and then grabs SES_old.

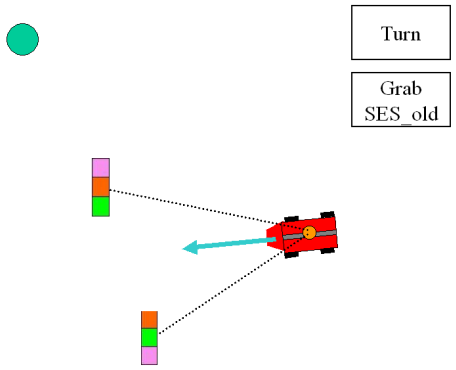

Figure 75. Turn and Grab SES_old

Now the robot moves forward a set distance a grabs a new SES. This is shown in the next step.

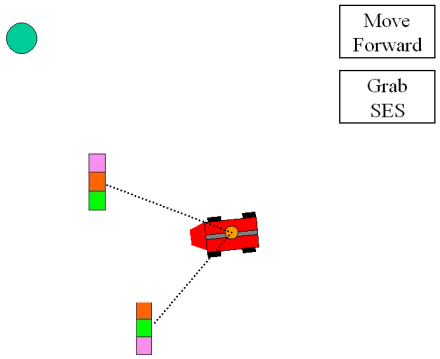

Figure 76. Moving and grabbing new SES

The robot then recalculates error and decides to continue. Now the robot can estimate the heights using the SES, SES_old, and the known distance of travel. It then stores the height estimates. This time, the robot can run 3D ENav+ to simulate through to the estimated point of convergence.

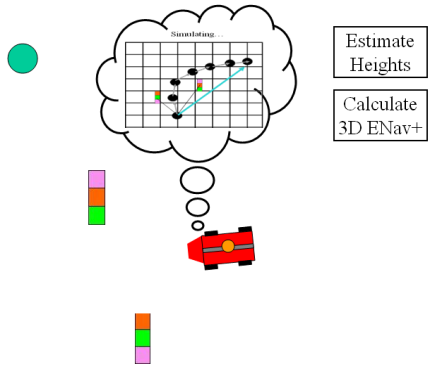

Figure 77. Simulating 3D ENav Iterations

3D ENav+ returns the direction of travel for the next iteration.

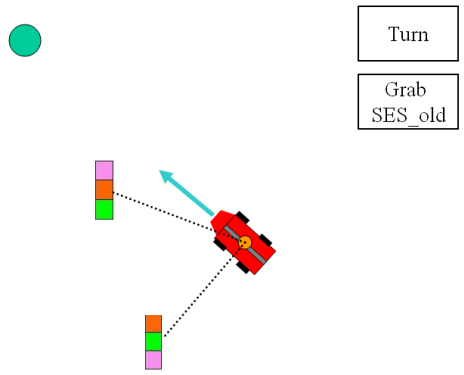

Figure 78. Turn, Grab SES_old

Now the robot turns in the direction assigned by 3D ENav+. It then grabs the SES_old and then process repeats until the robot converges onto the goal.

At this point, the robotic system implementation of 3D ENav +, hardware and software is complete.

CHAPTER VI

EXPERIMENTAL RESULTS

Overview

The designed system was tested in three ways. First, each sub-system of the overall system was tested individually. Additionally, a simulated version of the robot was thoroughly tested in a variety of circumstances. Finally, the total system itself was tested.

Throughout the paper, the tests confirming the correct operation of a given system is included with the general design of that system. For instance, the procedure for and results of testing the visual system have been included in with the discussion of the design of the visual system. The exception being a system such as motor-control where testing merely involves demonstrating the robot turns when asked to by the controller. This is confirmed in the test of the actual robot, rather then individually.

Simulation

The most thorough testing of the 3D ENav system was done in simulation. In these simulations, the same software that will drive the robot is in place on a simulated robot. This enabled testing of the system on a large variety of scenarios.

As was demonstrated throughout the discussion of 3D ENav and 3D ENav+, the simulations confirm that the algorithm performs correctly and it accomplishes its goals. The simulation interfaces with the ENav_C class, providing the same data the physical system

would, carries out ENav's requests in a simulated environment and repeats until ENav_C returns low error. Using this, the correct performance of ENav_C is confirmed for all cases from one to six landmarks. Additionally, all the objectives for improvement were met, specifically:

1) The number of landmarks required for convergence was lowered from 3 to 2
2) Convergence is guaranteed for all cases above 2
3) Using 3D ENav+, the paths taken are optimal

The only failing case discovered in simulation is the scenario when the goal and landmarks are precisely collinear. In this case the robot localizes to the line containing all landmarks and goal. However, because in the physical system, the robot will not be able to discern multiple landmarks from the goal if they are collinear with the goal (one landmark will be behind the other), then this scenario would have failed anyway. Additionally, the probability of precise colinearity in the physical world is very low, but this is moot in light of the limits on perception.

Indoor Navigation Results

The final phase of testing was to test the robot itself. Several tests were run in an indoor environment to confirm that the physical robot behaves the same as the simulated robot in its ideal virtual world.

The tests were set up to mimic those done in the simulated world. Landmarks were placed, and a goal region decided upon. Then the robot went through the 3D ENav algorithm in an attempt to find the goal.

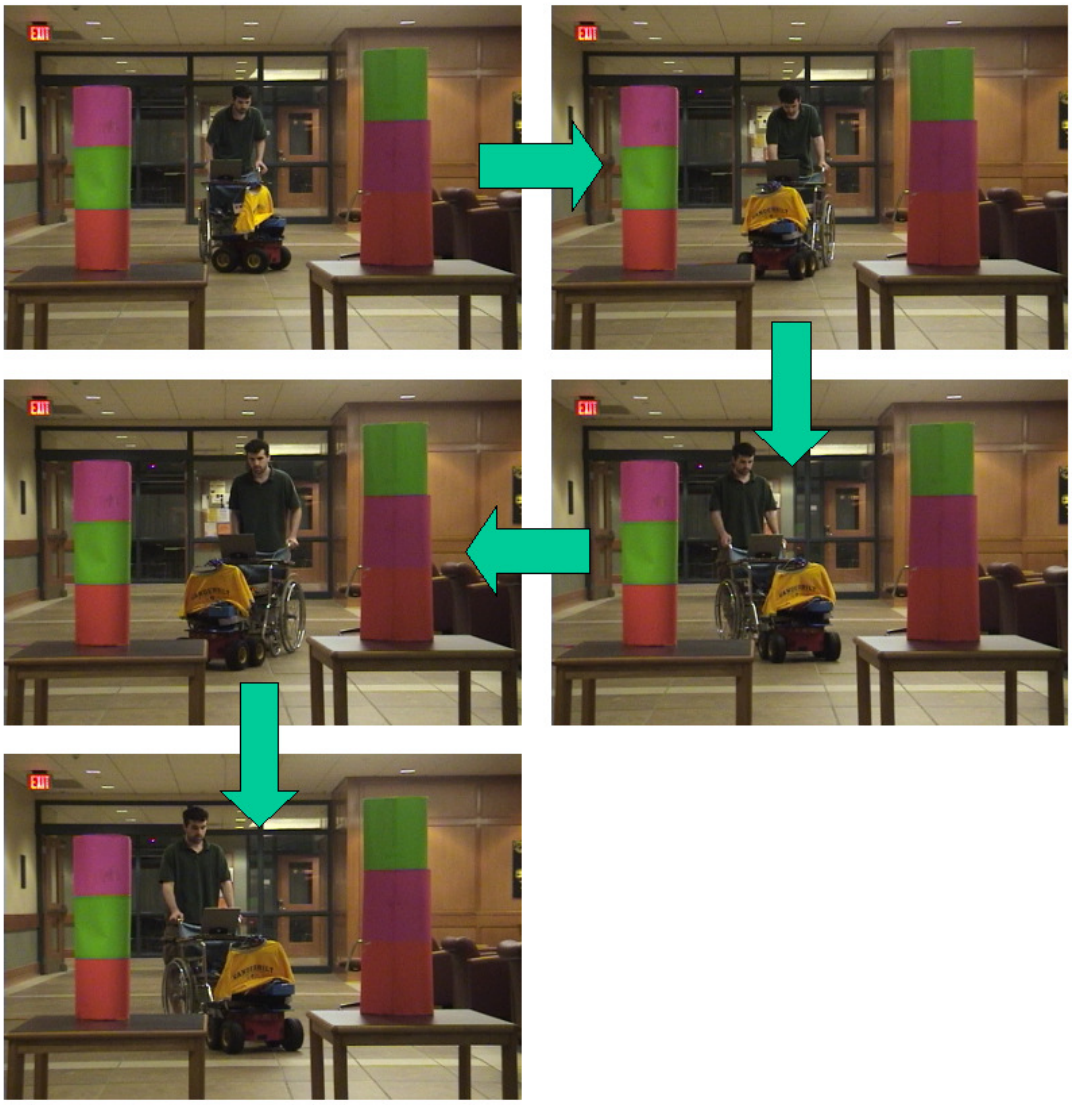

Figure 79. Experimenting in Indoor Environment

In the above figure, one test run of 3D ENav is outlined. The goal region in this example is at the center of the white line closest to the landmarks. It is shown that the robot zigzags toward the goal region. Because the robot in this example is run 3D ENav, and not 3D ENav +, this is exactly what is expected. This testing shows that 3D ENav in the

physical world behaves as it does in the simulated world. However, the actual robot suffers from a variety of problems not known to the simulated robot. Most acute among these deficiencies is its visual system. The camera system selected is highly inaccurate. Differences between cameras, blind spots, and frames lost due to constant switching between cameras severely impact the performance of the system. It is mainly because of this that although a successful 3D ENav run can be demonstrated, 3D ENav + is not possible with such inaccuracy. In the later recommendations section, solutions to this problem are suggested.

Overall, the indoor tests confirm that indeed the system does work, but it turns out that 3D ENav requires more accurate hardware then 2D ENav does. Essentially this is caused because 3D ENav's elevation data needs a higher degree of precession in order to make correct decisions.

CHAPTER V

CONCLUSION

In this paper, I have reviewed my design for a complete "go-to-goal" robot navigation system based upon previous work done at the Center for Intelligent Systems (CIS). Specifically, I have expanded on the 2 Dimensional Egocentric Navigation algorithm and from this expansion designed a complete robotic system for sensing, memory, navigational computation and actuation.

I have also demonstrated that this system is robust in its capabilities for goal finding, and accurate in path selection. The system requires no more information then what it perceives with its cameras (maps, GPS data etc.) and what it stores in short and long term memory (current SES and stored LES). It requires only two perceived landmarks for convergence on the goal, and will approach the goal in a direct manner. This result is essentially independent of the configuration of the goals and landmarks, barring unreasonable set ups, e.g. one landmark directly on top of another, etc.

CHAPTER VI

RECOMENDATIONS FOR FUTURE WORK

Regarding future work of 3D ENav and 3D ENav + there are a couple of possible research areas which could be pursued.

One possible direction could be to attempt to relax what is defined as a landmark. Other researchers in the CIS have been trying to develop methods of finding and recognizing naturally occurring landmarks such as trees as well as man-made objects already in place, such as exit signs.

In my implementation, the visual system acknowledges only very specific artificially constructed and placed landmarks. This was because the focus on the research was firstly to investigate a working 3 Dimensional Egospheric Navigational system and increasing the difficulty and unreliability of landmark detection would have been inappropriate at this point. However, with the system proven, relaxing the "visual constraint" of what defines a landmark will make the system more robust.

Establishing a higher level of control within the system is another direction of potential continued research. This would be along the lines of a hybridized system where 3D ENav+ would carry out behavior like low-level control, while a more deliberative system selects goals (LES's}. [Murphy, 2000]

There are a number of arbitrary parameters built in to the system, such as distance to travel each step, thresholds in visual processing etc. Another possible expansion of this

system could involve providing a method for the robot to tune these parameters itself, rather then relying on a human to set them beforehand. One method to accomplish this could be a to establish a critic [Russell, 2003] to rate the performance of the robot. The output of the critic could then be incorporated into a learning system that uses the rating to adjust the parameters over time. A neural network is one possible way to implement this.

Finally, by improving the quality of the equipment in the system, vast improvement can be quickly made. The area benefiting most would the visual system. A pre-packaged rotating camera, rather then the multi-camera set up currently employed would certainly be faster and vastly more accurate.

REFERENCES

[Albus, 1991] Albus J.S., "Outline for a Theory of Intelligence", *IEEE Transactions on Systems, Man and Cybernetics,* 21:3 May/June 1991.

[Free Dictionary Dot Com] http://acronyms.thefreedictionary.com/UUV

[Hambuchen, 2004] Hambuchen, K.A. "Multi-Modal Attention and Event Binding in Humanoid Robots Using a Sensory Ego-Sphere", Doctoral Dissertation, Vanderbilt University, 2004.

[Hunstberger, 2001] Hunstberger, T. "Biologically Inspired Autonomous Rover Control", *Autonomous Robots*, 11(11), 2001, 341-346.

[Kawamura, 2001] Kawamura,, K. R.A. Peters II, D.M. Wilkes, A.B. Koku, and A Sekmen, "Toward Perception-Based Navigation Using Egosphere", Lecture Slides, Vanderbilt University, Nashville TN, 2001.

[Kawamura, 2002] K. Kawamura, A.B. Koku, D.M. Wilkes, R.A. Peters II, and A. Sekmen, "Toward Egocentric Navigation", *International Journal of Robotics and Automation,* 17:4, November 2002.

[Kortenkamp, 1998] Kortenkamp, David, R. Peter Bonasso and Robin Murphy. Artificial Intelligence and Mobile Robotics. Menlo Park, CA: AAAI Press, 1998.

[Koku, 2003] Koku, Bugra "Egocentric Navigation and its Applications", Doctoral Dissertation, Vanderbilt University, Nashville, TN 2003.

[MATLAB] MATLAB Help Files

[Merriam Webster] http://www.dictionary.com/

[Murphy, 2000] Murphy, Robin R. Introduction to AI Robotics. Cambridge MA: MIT Press, 2000.

[Nell, 2002] Dale, Nell, Chip Weems and Mark Headington. Programming and Problem Solving in C++. Sudbury, MA: Jones and Bartlett Publishers, 2002.

[Peters, 2001] Richard Alan Peters II, Kimberly E. Hambuchen, Kazuhiko Kawamura, and D. Mitchell Wilkes. "The Sensory Egosphere as a Short Term Memory for Humanoids", *Proc. of the IEEE-RAS International Conference on Humanoid Robots,* Waseda University, Tokyo, Japan, 451-459,2001.

[Pinnete, 1991] Pinnete, Brian, "Qualitative Homing", *Proceedings of the 1991 IEEE International Symposium on Intelligent Control,* Arlington, Virginia, August 1991.

[Russell, 2003] Russell, Stuart and Peter Norvig. Artificial Intelligence: A Modern Approach 2nd Edition. Upper Saddle River, NJ: Prentice Hall, 2003.

[Tschumperlé, 2003] David Tschumperlé http://cimg.sourceforge.net/

Dr Wilkes, Casual Conversation