Navigation by Isolated Landmarks

Il-Pyung Park
Department of Computer Science
Columbia University
New York, NY 10027
CUCS-015-92

April 1992

Abstract

In this paper we propose and demonstrate a simple method for navigation in a large unstructured environment that contains featureless objects, which uses "isolated" landmarks in the navigator's view.

The map-maker and navigator are implemented using an IBM 7575 SCARA robot arm, the PIPE (Pipelined Image Processing Engine), and two cameras. The navigational environment consists of a flat plane with spherical objects populated randomly on it. First, the map-maker model observes the environment, given a starting position and a goal position, and it generates a "custom map" that describes describes "how" to get from the starting position to the goal position. The accuracy and the efficiency of the directional instructions are then demonstrated by the navigator by following the commands in the "custom map".

This is a first step towards our eventual goal, which is to develop a full set of vocabulary that can qualitatively "describe" the navigational environment to direct the navigator with efficiency and accuracy.

1 Introduction

Navigation in a large unstructured environment requires different information and tools than that of navigation in a small structured environment. The information that is useful is mostly qualitative data instead of the quantitative ones. Therefore, we feel that high-level reasoning is necessary for the navigator to be able to handle ambiguities and errors. By high-level we mean invariants that can be used to capture the appropriate navigational information. For example, "follow this road until you see a big white building on your right", is a good directional instruction, whereas "go straight for 200 meters and turn 40 degrees clock wise, ..." is not. One of the most important entities in giving directions is the landmarks. Without landmarks, it would be impossible to give "high-level" directional information. It would also be very hard for the navigator to verify that it has reached its destination. In this project, we will explore the nature of landmarks. We will attempt to answer important questions such as, "what is a good landmark ?", "what features of a landmark are important ?", and "what sensors are to be used to recognize a landmark efficiently ?".

One of the earliest work done on robot navigation based on human psychology in spatial reasoning was by Kuipers[Kui78]. He adopted the concept of "cognitive map" in which the spatial knowledge is divided into a hierarchy of four levels, namely, "sensorimotor", "procedural", "topological", and "metrical". In general, assimilation of knowledge (learning the map) proceeds from the lowest to the highest level, and the higher the level, the more complex cognitive computation becomes. Matching the spatial knowledge with the current environment (navigating) would involve the same semantic hierarchy and complexity. So, for example, trying to match "topological" knowledge with the topological features of current environment involves a greater cognitive complexity than trying to match "sensorimotor" knowledge with the features of current environment. Verbal direction is basically a "sensorimotor" or a "procedural" description, whereas visual map is "topological" and "metrical". When navigating using "topological" or "metrical" information, one has to somehow convert the information into "sensorimotor" or "procedural" knowledge because the latter information is easier to match with the environment that one is navigating.

The idea of giving directions in terms of "sensorimotor" or "procedural" information is supported by the result of the investigation by Streeter et al. [SVW85]. They performed an experiment on a group of people to compare the effectiveness of four different types of navigational aids: vocal instructions, simplified maps, both together, and commercial maps. The result revealed the superiority of vocal instructions over the other three methods. This result is somewhat expected since driving is an intensive task which requires the driver's constant visual attention, whereas his or her auditory channel is less busy. It is interesting to note, however, that using both vocal instructions and simplified maps was less useful than using only vocal instructions. This indicates that too much information degraded the performance of the drivers. The investigators also emphasize that the properties (qualities) of voice instructions that they provided, which were derived from a careful study of navigation related psychology, played an important role. The key point here is that it is very important not to over-burden the navigator with too much information.

Our aim is to build a map-maker that derives the most effective "sensorimotor" and "procedural" descriptions, using mostly "topological" information of the environment, that will lead the navigator to the goal position with both efficiency and accuracy. In other words, we are looking for the "ultimate short-hand" of directional instructions.

	Map-maker	Navigator
Objective	Generate a custom map	Navigate using a custom map
Visibility	Infinite	Limited to current view window
Metric ability	Yes	Only within current view window
Intelligence	Omniscient	Limited to interpreting the custom map
Memory	Large	None
Computing power	Fast	Slow

Table 1: Capabilities and limits of the map-maker and the navigator

2 Definitions

2.1 Assumptions

There are two major modules in this project, namely, the map-maker and the navigator. The terms - the map-maker, the navigator, the custom map - that are used in this paper were originated from [KL89]. The assumptions that we make in these modules are presented in this section.

The purpose of the map-maker is to generate a "custom map", consisting of directional instructions, so that the navigator will be able to find its way to the destination by executing each of the instructions in a sequence. The map-maker is assumed to be omniscient and error free. It sees the whole environment and knows the exact position of each objects that exists. The map-maker also knows the ability and the limits of the navigator. Therefore, the "custom map" contains directional instructions that the navigator can handle. The communication between the map-maker and the navigator is done off-line (one-way, one-time). This means that the custom map is given to the navigator in the beginning of the journey and the navigator's only source of directional information is the custom map.

The navigator's capabilities are much more limited. Its view window size is very small relative to the environment. It has limited metric measurement capabilities only within its current view window. However, the accuracy of the its metric measurements is assumed to be low. For example, the navigator can move the window position so that a visible reference object is positioned near one of the four corners of the view window, but it does not have the accuracy to pin point the coordinates of a visible object. Therefore, it is not possible for the map-maker to give, to the navigator, the (x, y) coordinate of a landmark as part of the directional instructions. The navigator is not intelligent enough to decide on its own what it should do, thus totally dependent on the "custom map" that it is given. It does not have any memory. Therefore, "custom map" can be considered as the navigator's intelligence. Table 1 summarizes the assumptions that we make on the map-maker and the navigator.

2.2 The World, the map-maker, and the navigator

We distinguish three similar but subtly different perceptions of "the world".

The real world The three-dimensional world as it exists and which is experienced by both the map-maker and the navigator. This is the real world and it is mathematically rich: it is continuous, it has a distance measure, and objects "embedded" in it can have finite extent. Much of this appears to be extraneous to the navigator: many such worlds consist largely of the essentially empty spaces between objects. Therefore, we need to abstract this world into something more simple.

The world model of the map-maker The first level of abstraction is done by the map-maker who is omniscient and error-free, into a graph. That is, the world is conceived as a collection of nodes which represent landmarks, connected by arcs which represent navigable regions without intervening objects. Empty space, distance, and object extent are ignored. Actually, these can be and are considered in the cost matrix by the map-maker. The data structures used in this level of abstraction are *parkways* and *trajectories*[Par91a].

The custom map The next level of abstraction occurs when the map-maker communicates (one-way, one-time) with the navigator in the form of a sequence: this is, by some criterion, the best "custom map"of visual landmarks, ordered by sequence of traversal. We postulate that this "custom map" is the optimal amount of information that the navigator needs in order to navigate. We assume throughout that there is only one navigator, and that the map-maker's omniscience extends to a perfect model of the navigator's sensory endowment. The navigator perceives the world in the most limited way. Sensory range is limited, to a relatively small area of the world (view window), and there is virtually no memory. However, this is an actual advantage; usually, the less that needs to be sensed or remembered the better, since the cognitive load of the navigator is minimized [SVW85]. Although more intelligent navigators can be modeled, this one chooses to ignore most of the world and its features for the sake of efficient traversal. Formally, the custom map can is represented in the form: $(D, M)^*$. D is a qualitative description to distinguish a landmark(s) from its neighboring objects. M is the movement of the robot in the local coordinate using the chosen landmark(s) as the reference point. For example, the *goal* object in figure 1, which is the dot located at the top, can be achieved by a custom map as follows: *((*,SW), (isolated,SW), (obvious,SE), (new,SE), (isolated pair,SW), (isolated,*))*. Here, the symbol $*$ is the wild card that matches anything. The initial position of the robot is indicated by the box at the bottom of the diagram. The first instruction of the custom map says "choose any object visible and move the view window so that it becomes positioned at the south west corner of the view window". The result of the robot's movement puts the robot in the position indicated by the next box (second from the bottom). Here, the robot sees many objects, but since the next command is *(isolated,SW)*, the robot chooses the isolated object and moves to the next position. The rest of the navigational path instructed by the custom map is depicted likewise.

Designing the "description language", such as *isolated, obvious, new, etc.*, to be used in the custom map and generating the custom map using such a language are some of the main research topics of this thesis project. The "description language" will be discussed in detail later in this paper (see page 8).

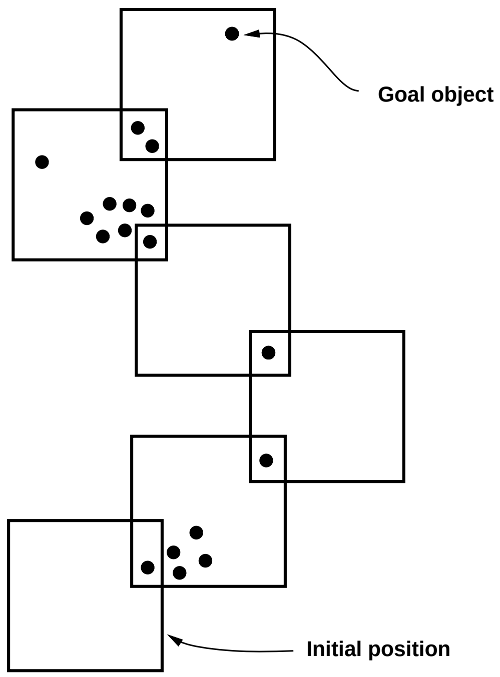

Figure 1: An example of navigation using a custom map

2.3 The navigable world

The navigational environment that we are interested in is a simplified three dimensional world. The navigational terrain itself is a flat surface that is visually uniform all over. Objects that will be scattered over this flat surface are spherical objects, such as marbles, uniform in size. There is no restrictions on where the objects are placed, except that no objects are allowed to be placed on top of another. The two assumptions of the objects - that they are uniform in size, and that they are placed randomly - bring up some important issues that were not present in our earlier project with the dots world[KPY90]. Firstly, an object can no longer be described by its intrinsic attributes, such as shape, size, or color (compared to this, our earlier project had 511 distinctly shaped objects). Therefore, in order to describe an object, the geometrical relationship of the target object to its neighboring objects must be considered. Secondly, the directions of the movements of the navigator is no longer restricted to a grid-like environment, thus serving as a more realistic model for robot navigation. Our goal is to be able to throw a number of marbles on a table and have our robot successfully navigate through this random world.

2.4 Robot movement capability

Ultimately, our navigator robot will have 4 degrees of freedom (x, y, z, θ): (x, y) to move to a desired (x, y) position in the random world, z to control the extent of visibility, and θ to look around its current position (pan). For simplicity, the robot navigator will not be given the freedom to control the angle of tilt of the camera. It will therefore be allowed to view the world only in a constant angle. However, the current version of the navigator has 2 degrees

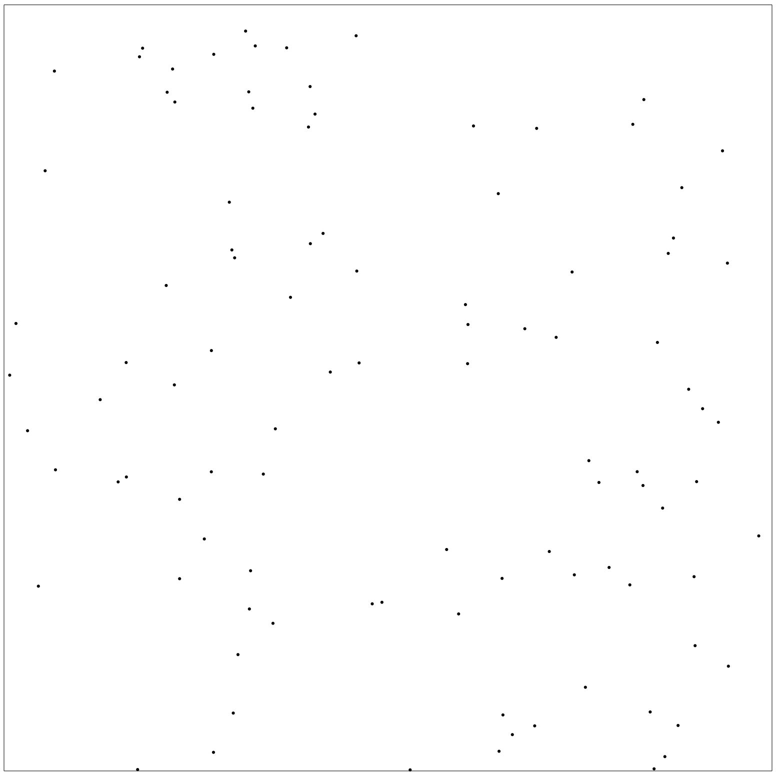

Figure 2: Randomly populated environment

of freedom in in the direction of x and y. Others, i.e. z and θ, will be fixed to constants until at later stage.

2.5 Parkways

First, as stated earlier, the map-maker abstracts the real world into meaningful data structures. After capturing the entire world by taking the image of the world from a vertically high location, the positions of the populated objects are recorded. Figure 2 shows an example of a randomly populated world with 100 objects.

The map-maker is assumed to know in advance the capabilities of the navigator, such as view window size and degrees of freedom. The map-maker further abstracts the world into a *graph* data structure, with vertices and edges. There are many ways to decide whether or not two vertices (objects) in this graph are connected by an edge. One way is to define that

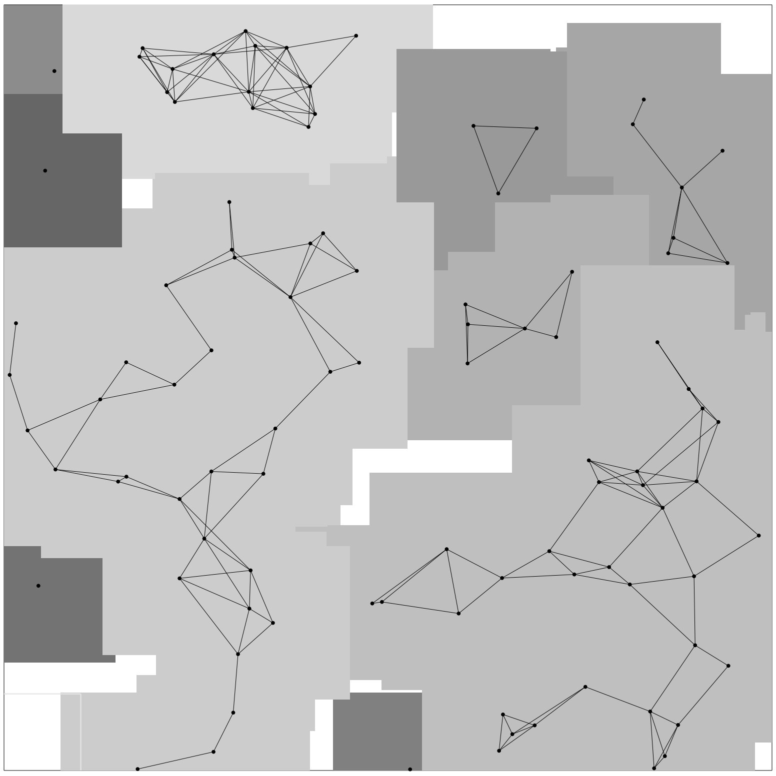

Figure 3: *Parkways*

two objects are "connected" if the two objects can be viewed in the same view window of the navigator. By applying the connected component algorithm using the above definition of "connectedness", we can then generate discrete sets of connected components. We define *parkway*[1] to be such a connected component. Figure 3 shows *parkways* formed on a world of 100 objects using a window size 10×10 in a world of size 100×100. The relative size of the window is shown in the lower left corner of the figure. The arcs between points indicate that the objects are connected. The regions with various shades indicate the extent of each of the *parkways*. These regions are defined as *territories*. When a new object is added to the environment, it is absorbed into the *parkway* corresponding to the *territory* where the new object is placed.

To find the shortest path between two objects in the same *parkway*, we can apply Dijk-

[1] *Parkway* will be sometimes referred to as *all parkway* when there is a need to distinguish it with other types of parkways (e.g. *obvious parkway*, *isolated parkway*).

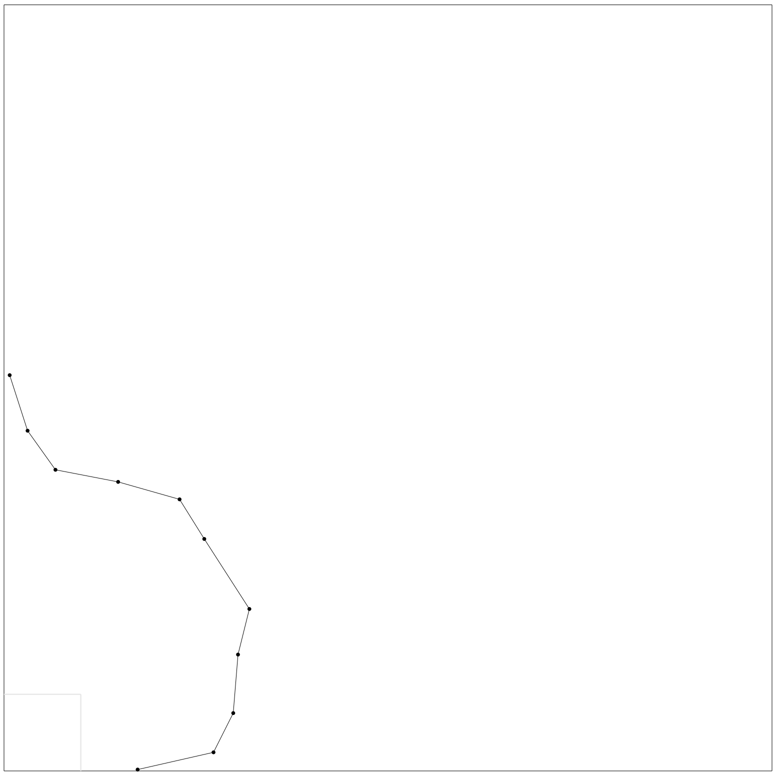

Figure 4: Shortest path within a same parkway

stra's shortest path algorithm. The candidates of next node to choose is confined to those objects which are connected to the *current* object. The *current* object is an object within the view window that the map-maker and the navigator use as a reference point. This way, we can guarantee that the robot does not move via metric information outside its sensible region, thus eliminating the possibility of a dead reckoned error. Figure 4 shows an example of the shortest path between two objects that are in the same *parkway*.

2.6 Description Language

The map-maker needs to generate a custom map that describes how the navigator may follow the landmarks along the computed shortest path. In this section, we explore the issues in designing the "language" for the custom map.

At any given instant, the navigator will have only a small portion of the world at its view, which may contain several objects. The map-maker model has to be able to “describe” what the navigator sees in order for the navigator to be able to distinguish a particular object to use as the next reference point. This is a hard problem because of our assumption of the navigable environment which is comprised of point-like objects that are randomly placed. Assuming that the robot has infinitesimal accuracy, infinite memory and extremely fast processor, it could keep the bitmap image of each possible view, or at least the coordinate of each landmark within each possible window. But realistically, we need some invariants, such as colors or shapes, to describe the immediate environment. Since our world is a “monochrome / point-like” world, we need a qualitative language that can describe the shapes of geometrical relations of the visible objects. In other words, we need qualitative and concise descriptions to represent “what you see”. The level of detail in the description process would depend on the intelligence and the mobile ability of the navigator.

The simplest non-metric method of discriminating an object would be to enumerate all the objects from left to right, top to bottom. But this method is very sensitive to the orientation of the camera. If the camera is even slightly misaligned to the original direction, this method would fail.

A slightly more robust method would be to enumerate the objects in a circular order. This would involve a “wrapping around” algorithm, similar to the “gift wrapping” method to derive convex hulls. This method, let us call it *spiral inward* method, is not susceptible to the errors in the rotational angle of the camera that can be critical to the *linear order* method. A *spiral inward* method starts at the *current object* and selects the next object by rotating an imaginary line clockwise until it hits a new object while pivoting one end on the *current object*. This process is repeated from each of the selected object until every object is touched. The starting angle of the line in the beginning can also be encoded. This could eliminate some ambiguities of which object gets selected first. For example, if *current object* is in one of the corners (N, E, S, W), then a possible starting angle would be 0, 90, 180 or 270 degrees. Time complexity of this method is $O(k^2)$, where k is the number of visible objects.

A more general method of scene analysis would involve a two step (large scale, small scale) process. In a large scale, computationally determining the clustering of objects would help to narrow down the “candidates” and is a research area by itself. Once the objects have been divided into clusters, the next stage can describe the individual clusters. Known methods for describing individual clusters, or *dot patterns* or *dot figures*, include *external shape* description method in [EKS83], *internal shape* description method in [Rad88] [Tou88], and fuzzy logic classification in [Abe91].

Here, we introduce some of the basic vocabulary that we have so far developed as part of our description language. An object is defined to be *obvious*, if it is the only other object that is visible except for the reference landmark. Two or more objects are defined to be *confusable* if it does not matter which object is to be chosen as the next reference point. The result of the next move of the navigator using any one of the *confusable* points should get the navigator to the same place. The term *new* objects refers to the objects that newly came into the view window as a result of robot's movement. The term *isolated* point refers to a single isolated object as a result of applying a clustering algorithm. The term *isolated pair* refers to a single pair of isolated objects as a result of a clustering algorithm. Figure 5 illustrates these methods.

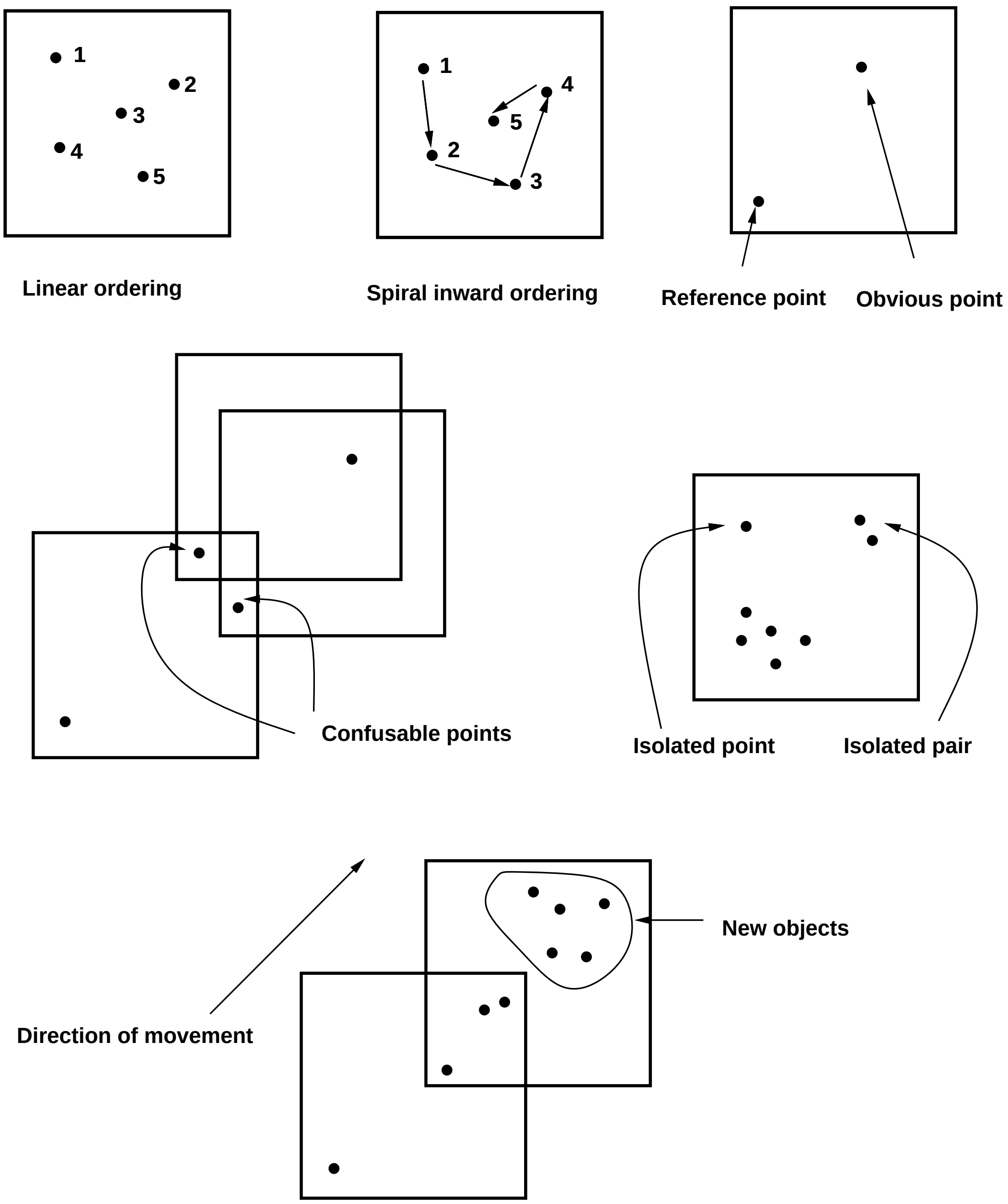

Figure 5: Examples of description language

3 Implementation of Isolated Landmark Following

In this section, we demonstrate a navigational technique using one of the simplest vocabulary in the description language - the *isolated point.*

3.1 Calibrating the map-maker

The omnisciency assumption of the map-maker requires that the image captured by the global camera (see Figure 10) correctly be used to generate the position of each object. To account for the image distortions due to translation, rotation, and perspective, we use a simple geometric calibration matrix A that transforms the homogeneous world coordinates $(X, Y, Z, 1)$ into the homogeneous camera coordinates $(U, V, 1)$[FGC87]. The transformation is represented by the following equation:

$$\vec{c_h} = A\vec{w_h} \tag{1}$$

where $\vec{c_h}$ is the homogeneous vector representation of the camera coordinates, A is the transformation matrix, and $\vec{w_h}$ is the homogeneous vector representation of the world coordinates. More precisely:

$$\begin{bmatrix} c_{h1} \\ c_{h2} \\ c_{h3} \\ c_{h4} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

The camera coordinates, U and V are then:

$$U = \frac{c_{h1}}{c_{h4}} \tag{3}$$

$$V = \frac{c_{h2}}{c_{h4}} \tag{4}$$

Substituting equation 3 and equation 4 in equation 2 and expanding the matrix products yields:

$$Uc_{h4} = a_{11}X + a_{12}Y + a_{13}Z + a_{14} \tag{5}$$

$$Vc_{h4} = a_{21}X + a_{22}Y + a_{23}Z + a_{24} \tag{6}$$

$$c_{h4} = a_{41}X + a_{42}Y + a_{43}Z + a_{44} \tag{7}$$

We do not expand the matrix product for ch_4 because it represents the z-coordinate of the camera. Substituting equation 7 into equations 5 and 6 yields:

$$a_{11}X + a_{12}Y + a_{13}Z - a_{41}UX - a_{42}UY - a_{43}UZ - a_{44}U + a_{14} = 0 \tag{8}$$

$$a_{21}X + a_{22}Y + a_{23}Z - a_{41}VX - a_{42}VY - a_{43}VZ - a_{44}V + a_{24} = 0 \tag{9}$$

Since there are 12 unknowns, and two equations, we need at least 6 data points to solve equations 8 and 9. We used 24 dots uniformly distributed on the workspace of the robot. The camera parameters (the elements of matrix A) were then estimated by solving equations 8 and 9 by least square.

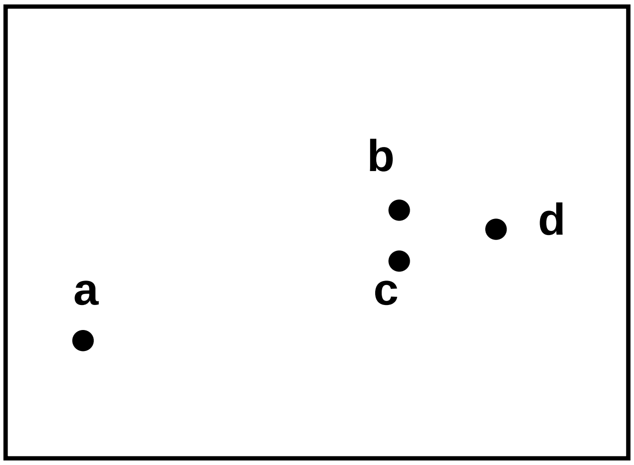

Figure 6: Finding the most isolated point

	a	b	c	d
a	0	5	4	6
b	5	0	2	3
c	4	2	0	4
d	6	3	4	0

Table 2: *mnv* matrix

3.2 Calculating the Isolated Point

Our algorithm to compute the most isolated point in a scene consisting of n point-like objects uses the concept of *mutual neighborhoods*[GK78]. The algorithm is as follows:

1. For each pair of points in the scene compute the *mnv* (mutual neighborhood value). The *mnv* of two points a and b is the sum of two numbers, representing the order of how close b is to a and the order of how close a is to b relative to all the other objects. For example, if b is 2nd closest point to a, and a is 3rd closest point to b, then the *mnv* is 5. The result is stored in a $n \times n$ matrix where n is the number of visible objects. The objects in figure 6 have the *mnv* matrix of table 2.

2. For each column of the *mnv* matrix, find the smallest value greater than 0. This value is the *mnv* value between this particular object (that correspond to this particular column) and its "closest" neighbor. Call this value the *c-value* of this particular object. So in our example, the *c-values* for a, b, c, and d are 5, 2, 2, and 3 respectively.

3. The object (column) that has largest *c-value* is the most isolated point in the scene. In our example, a has the largest *c-value* and therefore it is the most isolated point.

If we restrict our definition of "connectedness" (c.f. 7) we can create *isolated parkways* that represent viable paths. Figure 7 shows the *isolated parkway* of the randomly populated world of Figure 2.

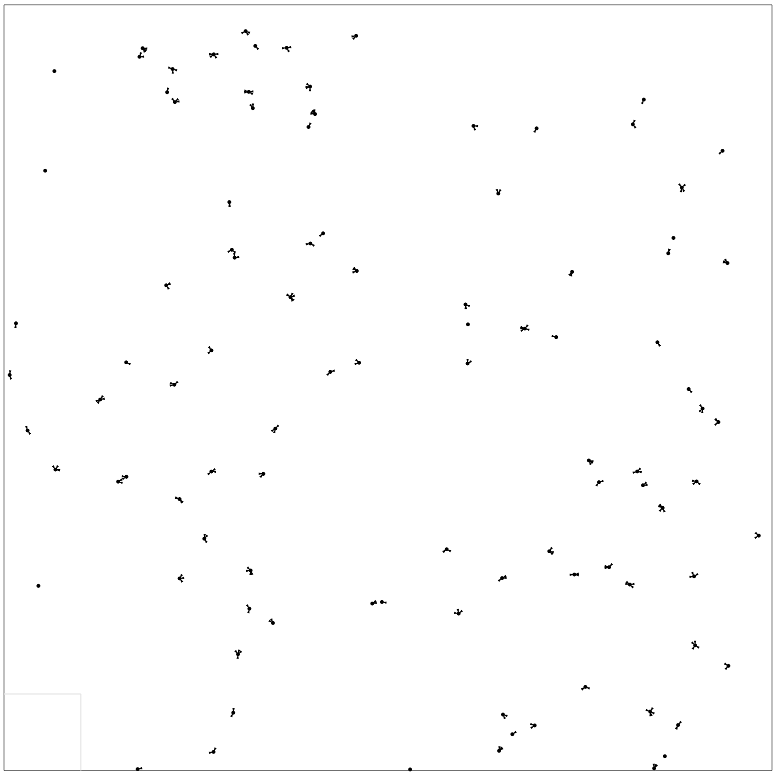

Figure 7: *Isolated Parkways*

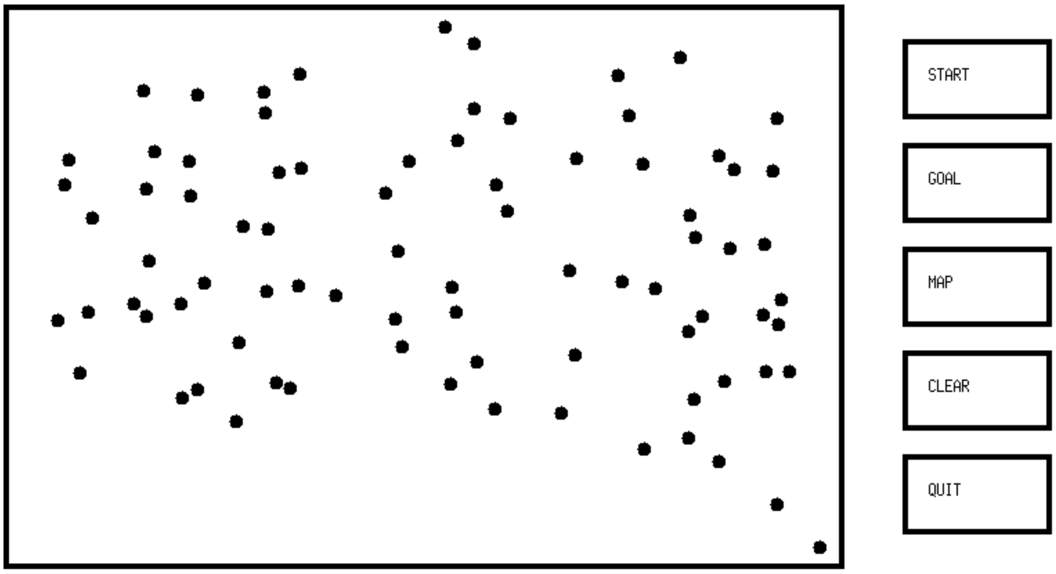

Figure 8: The user interface program for the map-maker

3.3 Generating the Custom Map

The map-maker module captures the global image of the populated world and records the position of each object (centroid). The populated world is display by the user interface program (See figure 8) running on the *X-window* system. When the start and goal objects are chosen, the map-maker computes the shortest path in terms of "the most isolated point" in each view window of the navigator. Then result of the shortest path computation is shown in figure 9. In the screen, the white object at the lower right corner is the starting point and the white object on the left side of the screen is the goal position. Each of the small rectangle represent the navigator's view along its path. For example, the rectangle in the lower right corner of the screen that contains the starting point is the initial view of the navigator. The line segments in each of these boxes connect the landmarks which are the most points in the robot's path. Note that the robot has to "detour" near the goal because the goal position has a very close neighborhood point. The custom map for the corresponding path is generated by the map-maker. In the example of figure 9 the corresponding custom map contains the following instructions:

- *(isolated,SE), (isolated,NE), (isolated,SE), (isolated,SE), (isolated,SE), (isolated,SE), (isolated,SE), (isolated,NE), (isolated,NE), (isolated,SW)*

In each instruction, *isolated* means to choose the most isolated point and the next arguement (such as *NE*) tells the navigator how to move. For example, if the instruction is *(isolated,SE)*, the navigator finds the most isolated point and adjusts its position so that it is at the south-eastern corner of the view window.

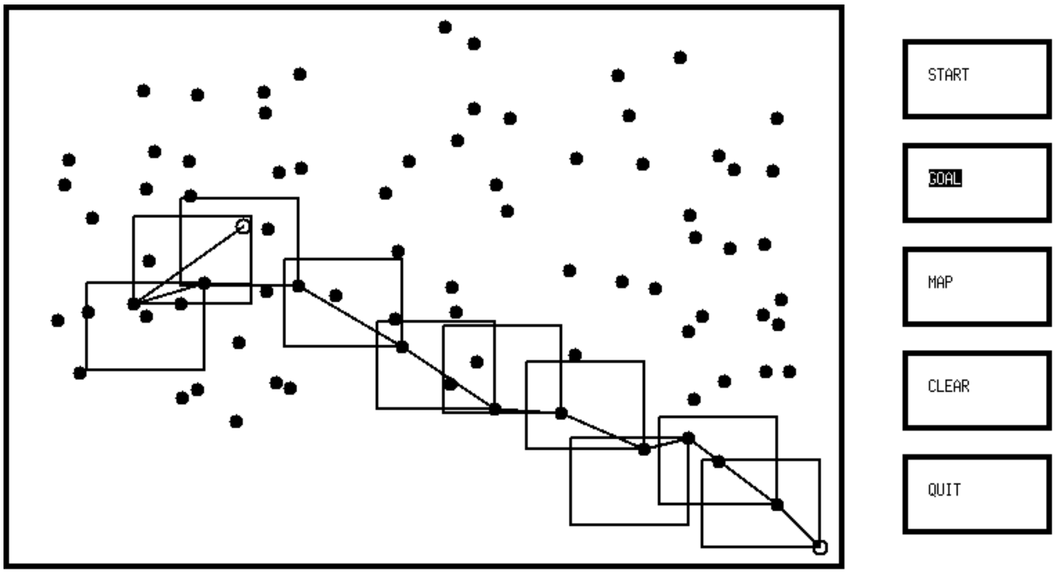

Figure 9: The computed shortest path

4 Results

We have implementing a simulated version of our map-maker and navigator using an IBM 7575 SCARA arm, two CCD cameras, PIPE, which is a high speed real-time image processor, and a SUN-4 workstation for high level control of the navigator and as the map-maker. Figure 10 and 11 shows the configuration of the map-maker and the navigator.

The map-maker is comprised of one CCD camera located at a position that can capture the whole workspace of the navigator. This camera is attached to the PIPE which grabs the image and sends it to the SUN-4 workstation which runs the "map-making" program based on the centroid information of the scattered objects. The map-maker generates a file called "custommap" which contains the list of directional instructions for the navigator (see page 14). The navigator is comprised of a second camera attached to the IBM robot arm. This camera is also connected to the PIPE for image processing of each scene as the navigator moves along. For each directional instruction in the custom map, the derivation of the most isolated point and the amount of "adjust" of the robot for the corresponding instruction is computed by the SUN-4 workstation. It then sends out low level instructions to the IBM arm controller for the actual movement.

The experiment was carried out for various types of population and start/goal positions. No quantitative method of evaluating the effectiveness of using isolated points for navigation exists at this time. However, we do have some empirical results. Firstly, the navigator tends to fail in subareas of the environment where the objects are highly populated. The navigator, whose is metric ability is poor sometimes fails in a highly cluttered areas because of mainly two reasons.

- Expected objects do not appear in the view window.

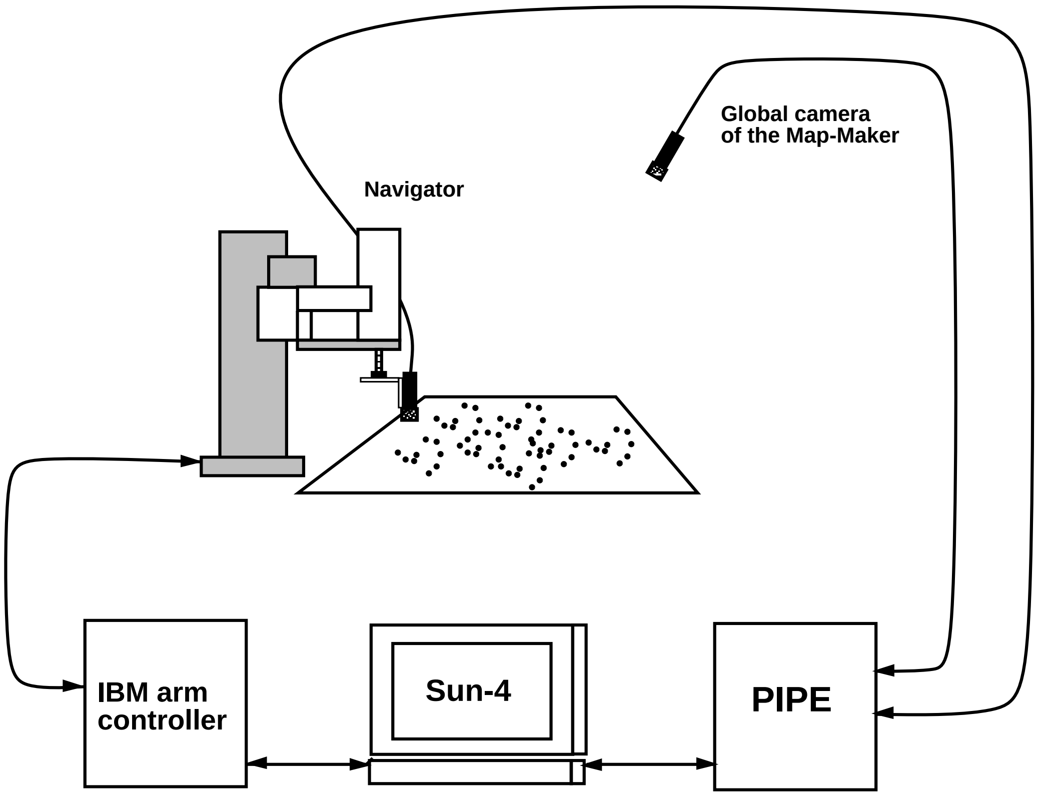

Figure 10: The experimental setup configuration

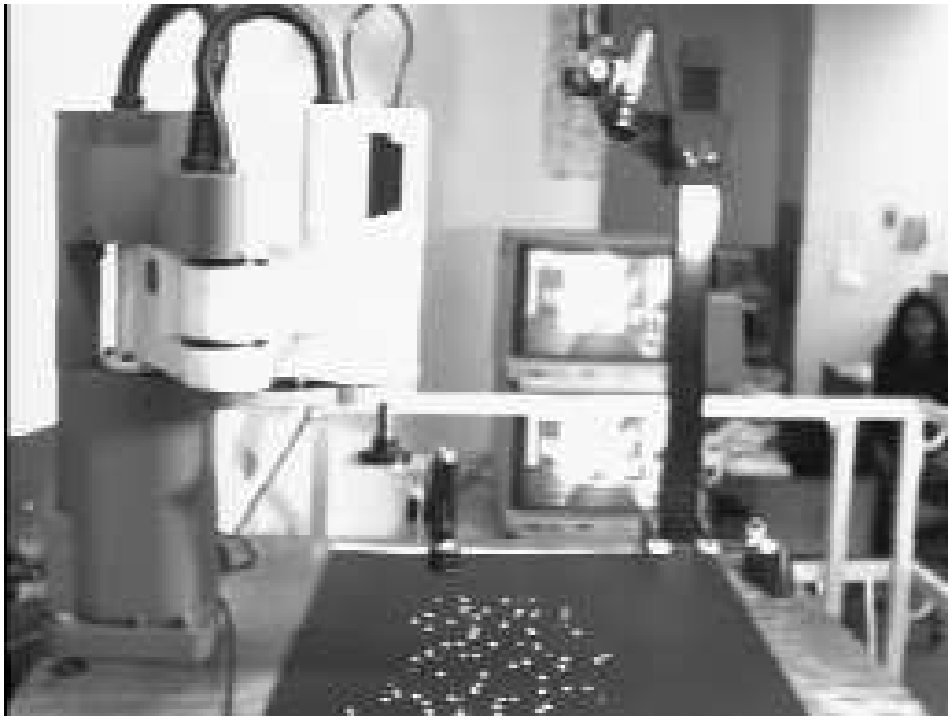

Figure 11: The map-maker and the navigator

- Unexpected objects appear in the boundaries of the view window.

Secondly, the navigation was more successful in sparse environment than in a densely populated environment. In a densely populated environment, there are many highly cluttered areas that increase the possibility of the navigator choosing the wrong object as the next reference point. Figure 12 shows an example of navigational path where the navigator crosses a “wall” of densely populated objects. The navigator is likely to become lost while crossing this “wall”.

We feel that there is a strong connection between the reliability of an *isolated point* and its likelyhood of becoming an *obvious point* (see page 9 for the definition). Since an *obvious point* does not have many objects near it, it is easier for the navigator to correctly identify it as a landmark. Therefore an *isolated point* is reliable if it is an *obvious point*. As the population density of the environment increases, the number of *obvious parkways* also increase, which means that the average length of navigable paths via *obvious points* decreases. This means that the reliability of *isolated* paths decreases as the population increases.

A simple heuristic to prevent the navigator from traveling along densely populated path is to “weight” each of potential isolated points by the number of objects that are close to it. Deciding the the size of this “reliability window” will be of a future research and will likely be based on statistical analysis of the population and window size. However, for the time being, we use the size of the navigator’s view window. In another words, for each of the potential landmark, we place a rectangle of the size of navigator’s view window centered at the potential landmark. Then count the number of objects within this rectangle. This number is multiplied into the cost matrix entry that contains the distance cost of achieving

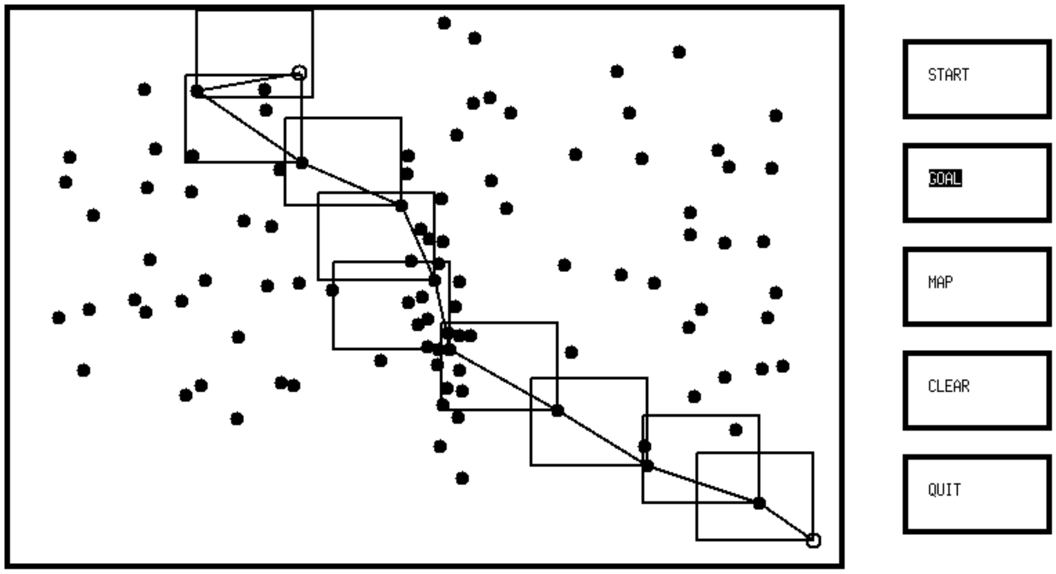

Figure 12: An unreliable path

this landmark. Since the more "neighbors" an object has, less likely it will be chosen as an *isolated* landmark. Figure 13 shows an example of navigational path that does not account for the reliability measures for *isolated points*. Figure 14 shows the result of the heuristic described above. Notice the navigator's path is detoured to avoid the highly cluttered area.

5 Conclusion

In this paper, we explored a method of guiding a robot navigator using *isolated points* as landmarks. Concept of *mnv* were used to compute the *isolated points* in the scene. The map-maker and the navigator were implemented using CCD cameras and a robot arm. Limits and drawbacks in using *isolated landmarks* were also studied. And finally, a heuristic to guide the navigator out of highly cluttered areas, in which *isolated landmarks* become unreliable, were presented. Future work will include a more elaborate description language that can fully represent the navigational scenes, statistical analysis to help decide on which "vocabulary" of the description language to use for each situation, and error recovery schemes for the navigator to avoid, detect, and to correct for errorful situations.

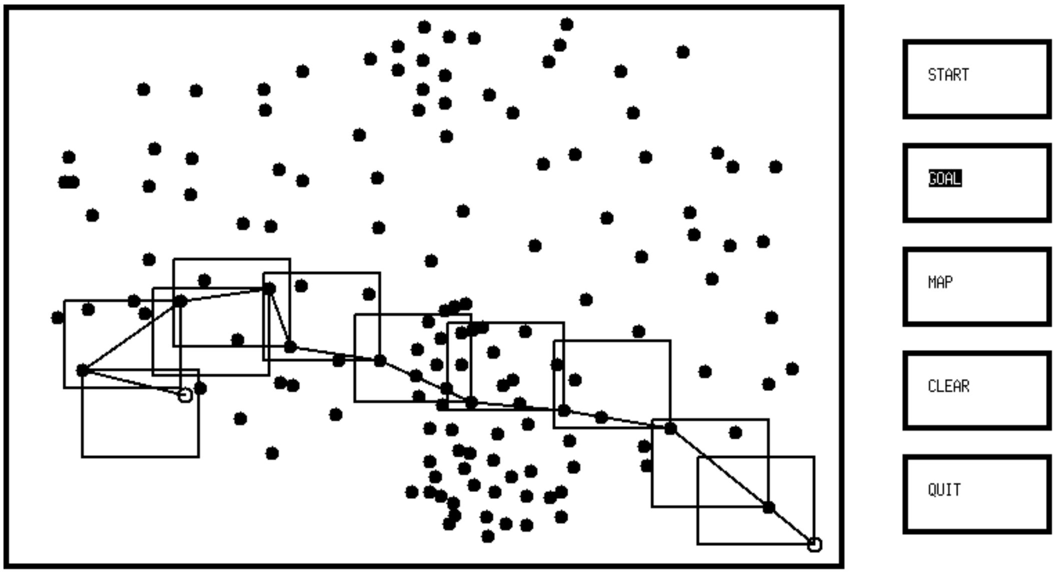

Figure 13: An unreliable path

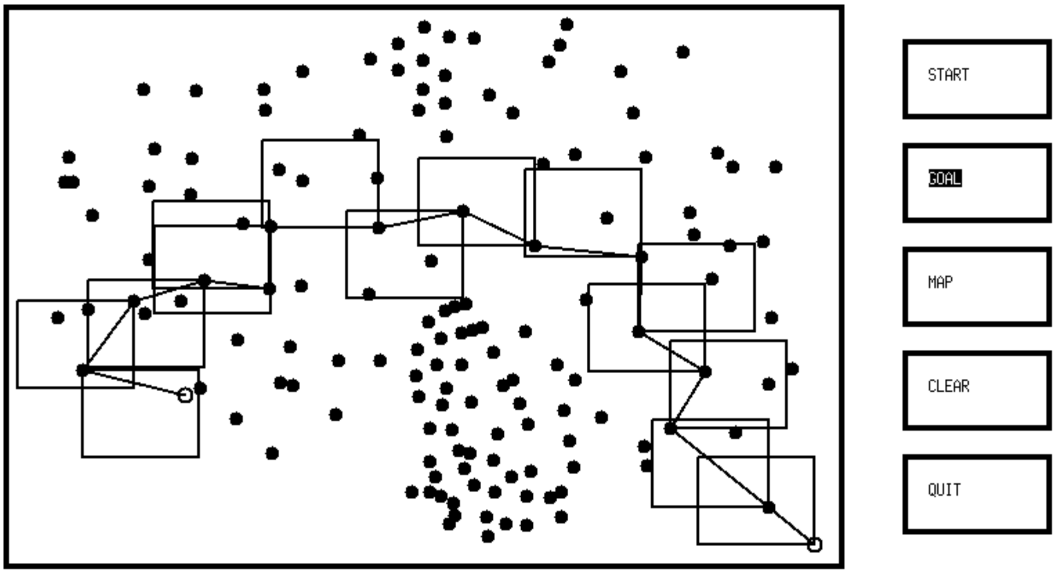

Figure 14: Result of heuristic to avoid highly cluttered areas

References

[Abb52] Edwin Abbott. *Flatland: A Romance of Many Dimensions.* Dover Publications, 1952.

[Abe91] Alica Abella. Extracting geometric shapes from a set of points. In *To appear in Proc. Image Understanding Workshop*, 1991.

[AHU74] Alfred Aho, John Hopcroft, and Jeffery Ullman. *The Design and Analysis of Computer Algorithms.* Addison Wesley, 1974.

[Asp87] Aspex Inc. *Introduction to PIPE Model 1 Systems*, December 1987.

[EKS83] Herbert Edelsbrunner, David G. Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *IEEE transactions on information theory*, 1983.

[FGC87] K.S. Fu, R.C. Gonzalez, and Lee C.S.G. *Robotics: Control, Sensing, Vision, and Intelligence.* McGraw Hill, 1987.

[GK78] K. Chidananda Gowda and G. Krishna. Agglomerative clustering using the concept of mutual nearest neighborhood. *Pattern Recognition*, 1978.

[IBM86a] IBM. *IBM 7575 and 7576 Manufacturing Systems Hardware Library Site Preparation, Installation, and Specifications, Part Number 70X8867*, 1st edition, August 1986.

[IBM86b] IBM. *IBM 7575 and 7576 Manufacturing Systems Software Library AML/2 Manufacturing Control System User's Guide, Part Number 67X1370*, 1st edition, August 1986.

[KL89] John. R. Kender and Avram. Leff. Why direction-giving is hard: The complexity of linear navigation by landmarks. *IEEE Transactions on Systems, Man, and Cybernetics*, 1989.

[KPY90] John R. Kender, Il-Pyung Park, and David Yang. A formalization and implementation of topological visual navigation in two dimensions. In *SPIE International Symposia*, 1990.

[Kui78] B. Kuipers. Modeling spatial knowledge. *Cognitive Science*, 1978.

[Par91a] Il-Pyung Park. Qualitative environmental navigation: Theory and practice - thesis proposal. *Columbia University, Department of Computer Science*, 1991.

[Par91b] Il-Pyung Park. Towards automatic vehicle navigation. Technical report, Columbia University, Department of Computer Science, Technical Report, 1991.

[Per11] Harry W. Perry. Some remarkable mechanical road guides. *Scientific American*, 1911.

[PS85] Franco P. Preparata and Michael I. Shamos. *Computational Geometry, An Introduction.* Springer-Verlag, 1985.

[Rad88] John D. Radke. On the shape of a set of points. In *Computational Morphology*. Elsevier Science Publishers B.V., 1988.

[Ric83] Elaine Rich. *Artificial Intelligence*. McGraw-Hill, 1983.

[Rie80] C. K. Riesbeck. 'you can't miss it': Judging the clarity of directions. *Cognitive Science*, 1980.

[SVW85] L. A. Streeter, D. Vitello, and S. A. Wonsiewica. How to tell people wherer to go: Comparing navigational aids. *International Journal of Man-Machine Studies*, 1985.

[TG90] C. Thorpe and J. Gowdy. Annotated maps for autonomous land vehicles. In *Proceedings: Image Understanding Workshop*, 1990.

[Tou88] Godfried T. Toussaint. A graph-theoretical primal sketch. In *Computational Morphology*. Elsevier Science Publishers B.V., 1988.