Topological visual navigation in large environments

Il-Pyung Park and John R. Kender
Department of Computer Science
Columbia University
New York, NY 10027
CUCS-001-94

Abstract

In this paper, we investigate a new model for robot navigation in large unstructured environments. Our model consists of two parts, the map-maker and the navigator. Given a source and a goal, the map-maker derives a navigational path based on the topological relationships between landmarks. A navigational path is generated as a combination of "parkway" and "trajectory" paths, both of which are abstractions of the real world into topological data structures. Traversing within a parkway enables the navigator to follow visible landmarks. Traversing on a trajectory enables the navigator to move reliably towards a target, based on shapes formed by visible landmarks. Error detection and error recovery routines are also encoded into the path segments. The optimal path is further abstracted into a "custom map," which consists of a list of directional instructions, the vocabulary of which is defined by our environmental description language. Based on the custom map generated by the map-maker, the navigating robot looks for events that are characterized by spatial properties of the environment. The map-maker and navigator are implemented using two cameras, an IBM 7575 robot arm, and a PIPE (Pipelined Image Processing Engine.)

1 Introduction

In this paper, we provide a new framework for topological navigation in a large unstructured environment, that emphasizes the use of qualitative, high-level, landmark-based directional instructions that minimizes a navigator's efforts and prevents mistakes. Part of this paper is motivated by questions concerning landmarks, such as, (1) what is a good landmark (2) what visual features of a landmark are important (3) what sensors are to be used to recognize a landmark efficiently [KL89] (4) how to describe a landmark to the navigator (5) how to detect errors and (6) how to recover from them. In attempting to answer these questions, we suggest a dichotomy of roles in the navigation system: a superior direction giver (map-maker) and a dumb direction follower (navigator).

Previous qualitative approaches to robot navigation include: the TOUR model of Kuipers[Kui78], the NX Robot by Kuipers et al. [KB88], Qualnav by Levitt et al. [LT90], inexact navigation by Sutherland et al. [ST93], Dai et al.'s "range free navigation", and the PV (Panoramic View) representation used by Zheng et al [ZT92].

Most of the work in qualitative navigation use the general definition of "distinctiveness" of features in the sensory readings to indicate landmarks. We argue that without detailed criteria for defining and selecting good landmarks, topological navigation is not possible. In this paper, we present qualitative methods to identify landmarks, as we present means to navigate using the selected landmarks.

2 Definitions and assumptions

The navigational environment that we are interested in is a three dimensional world, although the current implementation of the navigator (due to its restriction of degrees of freedom) makes the effective environment two dimensional. 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 - emphasize the spatial and topological problems of doing vision and navigation "in the large".

There are two major modules in this project, namely, the map-maker and the navigator. The ultimate objective 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).

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 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 it is totally dependent on the custom map that it is given. It does not have any memory. Therefore, the 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.

3 Abstractions of the world

3.1 Parkways and trajectories

When the map-maker captures the real world with its global sensor (in our case, a camera), it records all the positions of the populated objects. As stated earlier, we assume that objects are identically sized and shaped. Therefore, the intrinsic qualities of each object is not useful for navigation. The map-maker further abstracts the world into a *graph* data structure, with vertices and edges. Each vertex represents an object and each edge indicates a viable path. There are many ways to decide whether or not two vertices (objects) in this graph are connected by an edge. In the most general case, we define two objects are "connected" if the two objects can be viewed in the same view window of the navigator and if one object can be identified in reference to another. This is the central idea of the parkway concept: a given object in the window can serve as the current reference landmark, and another object in the same window can serve as the current "goal" landmark.

Then, by applying the connected component algorithm [AHU74] using the above definition of "connectedness", we can then generate discrete sets of connected components. We define a parkway to be such a connected component. It is navigated somewhat like crossing a stream by hopping from rock to rock: each point must be (visually) reachable from the previous one. The left diagram of figure 1 shows an example of parkways formed for a navigator whose windows size is 10×10 on a randomly populated world of size 100×100 with 100 objects. The arcs between points indicate that the objects are connected. The regions with various shades indicate the extent of each parkway. 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.

Traversal between two disjoint parkways is done by a simple method called the "trajectory" movement. For each pair of objects in the same view window, we compute the feasibility of reaching a landmark from another parkway by sliding the window in the direction formed by the two objects. Measurement of connectedness of the parkways (through trajectories) is derived by applying the strongly connected component algorithm on this directed graph [AHU74]. The right diagram of figure 1 shows an example of trajectories computed on our random world. The straight lines are the trajectories of the window in the direction of the sliding movement. One end of a line is attached to the window at the position to which the new object is expected to appear. The other end of the line is the object that this sliding window is seeking.

Depending on the definition of "connectedness", there can be many different parkway types, such as obvious parkway, isolated parkway, reliable parkway, error detectable parkway, error recoverable parkway, and etc. Trajectory types can be categorized similarly.

3.2 Description language

The map-maker needs to generate a custom map that describes how the navigator may follow the landmarks along the computed best path. At any given instant, the navigator will have only a small portion of the world at its view, and the map-maker has to be able to describe what the navigator sees in order for the navigator to identify a landmark. Since our

	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
DOF	0	2: x and y
Communication	Off-line	Off-line

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

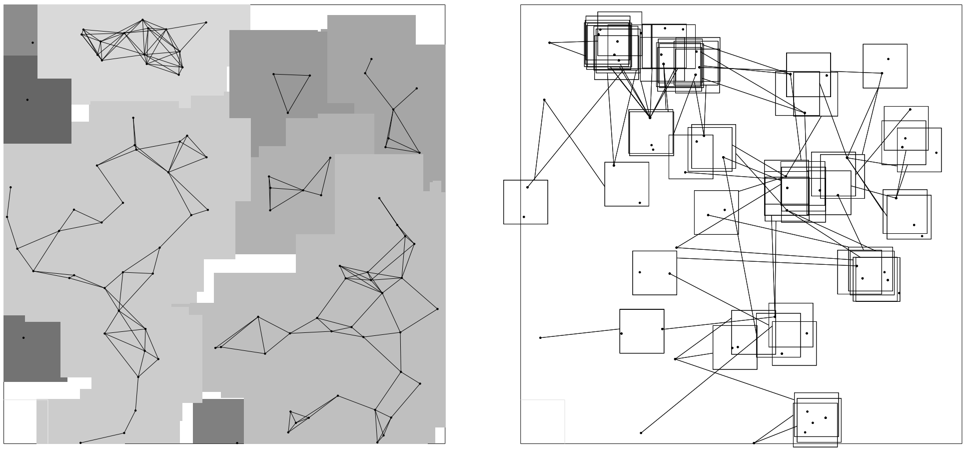

Figure 1: Parkways (left) and trajectories (right) formed on a randomly populated dots world

world is a point-like world, we need a qualitative language that can describe the geometrical relations of the visible objects. The level of detail in the description process would depend on the intelligence and the mobile ability of the navigator; our exploration here deliberately emphasizes the topological aspects of navigation. To our surprise, isolated landmarks were sufficiently robust and flexible, and therefore we were able to build a complete system by only using isolated landmark descriptors. Other types of landmarks that we considered include "obvious points", "isolated pairs", "confusable points", and "new points" [Par93].

4 Error handling

In this section, we focus on various problems that relate to errors that occur before and during the navigation. In the beginning of map-making session, the map-maker captures the entire environment with its global camera, and computes the real world (x, y) coordinates of the populated objects. The calculated coordinates contain error. Error is further accumulated by the navigator during the navigation due to its sensory and motor limitations.

There are two different ways to deal with errors. The first method is "error prevention" and is performed by the map-maker. Prior to the navigational task, the map-maker carefully examines the environment and generates a path that is least affected by error, thereby preventing the navigator from making mistakes. Analytic results, necessary for the creation of reliable directions, quantifying the robustness of parkways and trajectories under various perturbations such as sensor or movement error, are provided in [PK93] [Par93]. The second method is "recovery", and is accomplished by the navigator during its navigational task. At each "check-point", the navigator examines its immediate surrounding to verify that it is on the correct course. If an error has been detected, it carries out an error recovery routine to get back to the original course. In this paper, we discuss the error recovery method.

4.1 Error detection in parkways

When a landmark is obtained, the navigator needs to observe its neighborhood in order to verify that it is on the correct course. Usually, the current view of the navigator does not contain enough information for verification. The largest region that the navigator can observe without losing its current reference point is the area covered by the 4 quadrants - SE quadrant, SW quadrant, NE quadrant, and NW quadrant. When the reference object is at the SE corner of the window, the corresponding area covered by the window is the SE quadrant, and so forth. Mistakes or errors are detected by observing the 4 quadrants relative to the currently achieved landmark.

A landmark can be characterized by the contents of its 4 quadrants. The contents of the 4 quadrants can be expressed in a *signature* form of:

$$\textit{signature} = \{\text{Observe(SE)}, \text{Observe(SW)}, \text{Observe(NE)}, \text{Observe(NW)}\}$$

where Observe(Q) is a description of the corresponding quadrant $Q \in$ {SE, SW, NE, NW}. In the isolated landmark selection process, a wrong object may be mistakenly chosen. If the signature of the wrongly chosen object is different from the signature of the correct isolated landmark object, then the mistake can be identified by comparing the desired signature to the currently observed one. Therefore, in order for a landmark to be error detectable, it needs to have a unique signature within its neighborhood.

For our implementation and experiments, we used the Quadrant Occupancy Count (QOC), which represents the number of occupied quadrants[1]. QOC of a landmark is obtained by moving the view window around the 4 quadrants relative to the landmark and by counting the number of occupied quadrants. Errors can be detected only if the landmark has a unique QOC within the current view window. In figure 2, we see there are 5 objects in the view window. R is the current reference point and I is the desired isolated landmark. As we can see, QOC of object I is unique within this view window, thus error can be detected by examining the QOC of the obtained landmark.

[1] We suggest some other Observe(Q) functions, such as QOS, QPC, and QPS, in Park[Par93].

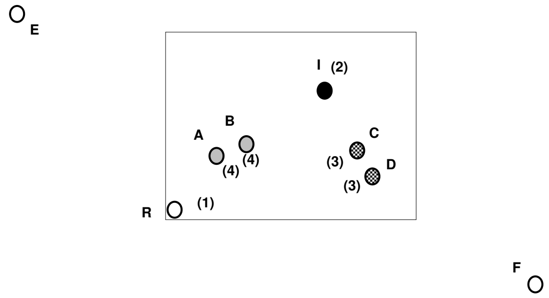

Figure 2: Error detectable configuration

4.2 Error recovery in parkways

When the error has been detected, the navigator is "lost", and therefore may be forced to give up its task of getting to the goal. However, in many cases, by performing a fixed set of "reset" sequences, the navigator is able to get back to its originally intended course. The basic idea is to assign a fixed action for each QOC (0 to 4). Therefore, the error recovery is a simple "if QOC then ACTION" set. For example, in figure 2, objects A and B have QOC of 4. If the navigator chooses either A or B instead of the desired I, it can "correct" the mistake by moving to the SW corner (relative to A or B) and then choosing the isolated point. Like wise, the correction instruction for objects C or D whose QOC is 3, is (SE, Isol).

Three conditions have to be satisfied for the error recoverability. First, the configuration has to be error detectable. Second, Objects in the same QOC group have to be located so that the originally intended landmark (the true isolated landmark, I) is in the same quadrant related to these objects. For example, in our example of figure 2, objects A and B having QOC 4, both have I in quadrant SW. Lastly, from every reachable object (which can be mistaken as the true isolated landmark), there must exist a path (directly or transitively) to the true isolated landmark. For example, in our figure 2, the isolated point of object C in SW quadrant is in fact object B, not I. But I is reachable from B. Therefore, I is transitively reachable from C.

4.3 Error detection in trajectories

We can use the QOC methods to detect errors in the trajectory movements by appropriately modifying the existing algorithm introduced earlier. In the parkway case, the landmark needed to have a unique QOC within the view window in order for it to be error detectable. In trajectories, the landmark (trajectory target) has to be unique among all the wrongly reachable objects.

4.4 Error recovery in trajectories

A trajectory movement is error recoverable if the configuration satisfies the three conditions similar to the ones described for the isolated landmark case.

Figure 3 shows an example of an error recoverable path. The initial position of the navigator is at the lower left hand corner. Initial reference object is the dark gray object. The goal object is near the top right hand corner, represented by the light gray object. The numbers next to the objects are the corresponding QOC's. Initially, the navigator sees 5 objects in its view. The first entry in the custom map instructs the navigator to identify isolated landmark and adjust to SW. If it makes a mistake and wrongly chooses an object whose QOC is 4, it is told to adjust to SW and look for an isolated landmark. If it chooses an objects whose QOC is 2, it is told to adjust to SE and look for an isolated landmark. Only when the chosen object's QOC is 3, it is able adjust to SW and continue to the next instruction. The next entry in the custom map instructs the navigator to do a trajectory movement based on directional vector formed by the reference point and the isolated point. If the QOC of the obtained landmark is 2, it is told to adjust to NE and to look for isolated landmark. Otherwise, if it is 1, it can adjust to SW and continue to the next instruction. The last entry in the custom map tells the navigator to identify an isolated landmark as the goal object. If the computed isolated point has QOC of 2, it is told to recover by adjusting to NW, and then by identifying the isolated landmark. Otherwise, if the QOC is 3, the goal has been identified.

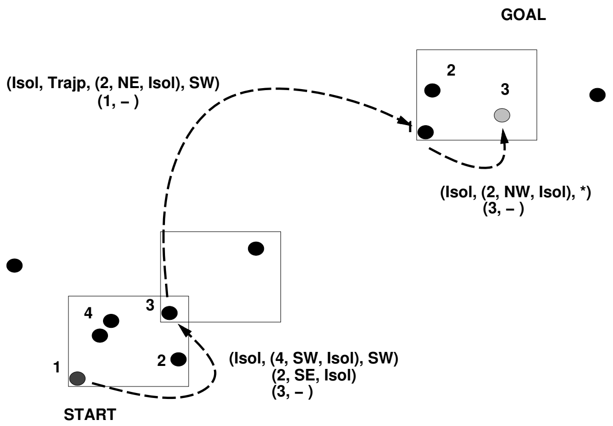

Figure 3: An error recoverable path

5 Path generation

The transition matrix contains information as to whether or not an object can be identified in reference to another. That is, if object B is directly reachable from object A, the entry of the transition matrix corresponding to (A, B) is set to 1. From the observation of the transition matrix, we note that an object whose corresponding column is filled most is a crucial landmark (thus, this is an operational definition of "good" landmark: something that is maximally reachable.) Each entry in the cost matrix, corresponding to the transition matrix entry, is a value indicating the cost of such transition. Depending on the need of the navigator, this value can bear different meanings. For example, this can be spatial distance, temporal distance, reliability measure, or any other index of evaluation that can be measured or generated.

5.1 Parkway and trajectory paths

The shortest path in a parkway is generated by Dijkstra's algorithm that searchs single-source shortest path to all other vertices. This requires $O(n^2)$, were n is the number of vertices in the graph. The overall shortest path is a combination of parkway path and trajectory path.

5.2 Definition of optimal path

Depending on the needs, different constraints can be imposed into the computation of optimal path. Some examples of optimality criteria are: (1) D - to minimize travel distance (2) R - to maximize probability of reaching destination (3) T - to minimize travel time (4) S - to minimize cognitive strain of the navigator involved processing its sensory inputs (5) M - to minimize number of entries in the custom map. Currently, we have implemented cost functions that estimate the distance of navigation path D and the unreliability of navigation path R, and (in a separate work [KPY90]) S and M of the 3×3 world[2]. One of map-maker's responsibility is to generate a path that either minimizes D, or maximizes R. Unfortunately, in some environments, these two cost estimates are in a direct conflict of each other (figure 4). For example, if the shortest distance path involves in a highly cluttered area, the reliability would be low. Conversely, a reliable path that avoids highly cluttered areas may

[2]This work did not include error detection/recovery, which would have substantially affected the communication cost.

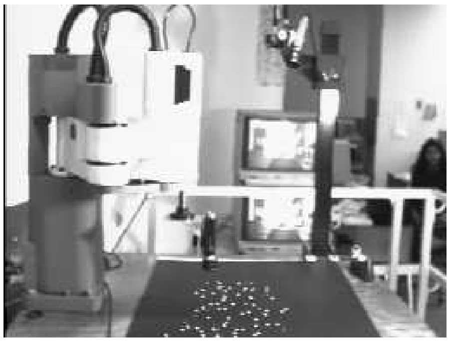

Figure 5: The experimental setup configuration

force the navigator to detour around a shortest path. We can see the difference between the D-path (leftmost diagram) and the R-path (middle diagram) in figure 4. Here, we suggest a third function C that compromises D and R, defined as: $C = \log D - \log R$. Using C as our cost estimate for travel, we can apply Dijkstra's shortest path algorithm to derive a path that minimizes the D/R ratio. The generated path will tend to favor short paths and sparsely populated areas. Note that R is a function of not only n (population) but also σ (position estimate error) [Par93]. If σ is small, the path will resemble the D-path and if σ is large, the path will resemble the R-path. This means that if the navigator's metric ability within its window is good, the optimal path will be the metrically shortest path. On the other hand, if the navigator's metric ability is poor, the optimal path will be the one that least confuses the navigator. An example of the C-path is shown in the rightmost diagram of figure 4.

6 Implementation and experiments

We have implemented our map-maker and navigator using an IBM 7575 SCARA robot arm, two Sony XC-77 CCD cameras, PIPE, which is a high speed real-time image processor, and a Sun SPARC workstation for high level control of the navigator and as the map-maker. Figure 5 shows the configuration of the map-maker and the navigator.

6.1 Sparse versus densely populated environment

Statistical experiments showed that the reliability of an isolated point is inversely proportional to the number of visible objects , n, and proportional to the sensor's accuracy, σ [Par93]. Our test runs with the robot arm navigator agreed with these statistical results. The robot tends to fail in (1) highly populated environments, and in (2) areas of environments where there are large clusters of objects. The left diagram of figure 6 shows a path created by the map maker that passes through the "wall" of a highly cluttered region of the environment. On the right, we see the adjusted path that avoids the unreliable cluttered region.

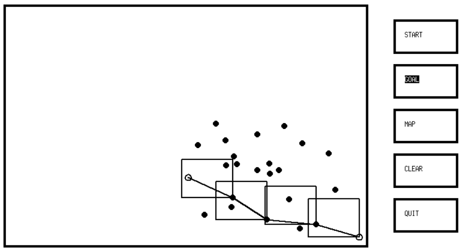

Figure 7: Example of a simple error recoverable path (path starts in the lower right).

6.2 Error detection and recovery

In figure 7, we see a path generated in a simple environment. The starting position is at the lower right-hand corner. Corresponding custom map for this path is:

((Isol, SE), (Isol, (4, NE, Isol), (3, -), SE),
(Isol, SE), (Isol, (4, NE, Isol), (2, -), -))

In our test run with the robot, the navigator failed to obtain the correct landmark during the execution of the second entry of this custom map (corresponding to the second window from the right in the path). The correct landmark was marginally out of the navigator's visual window and the navigator incorrectly identified the top-most object in the window as the isolated landmark. When the error verification sequence was executed, QOC was 4, which indicated the robot to adjust to NE and to recapture the correct landmark. In this case, the error recovery routine handled the error caused by the object which was positioned near the boundary of the view window. After the error recovery routine, the navigator was able to get back on track, and eventually get to the destination successfully. Effectively, the error recovery routine has increased the reliability of this path.

References

[AHU74] Alfred Aho, John Hopcroft, and Jeffery Ull-

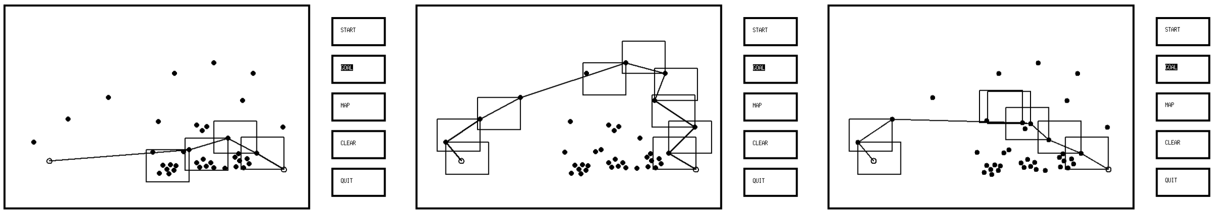

Figure 4: Shortest distance (left), the most reliable (middle) paths, and the compromised path (right)

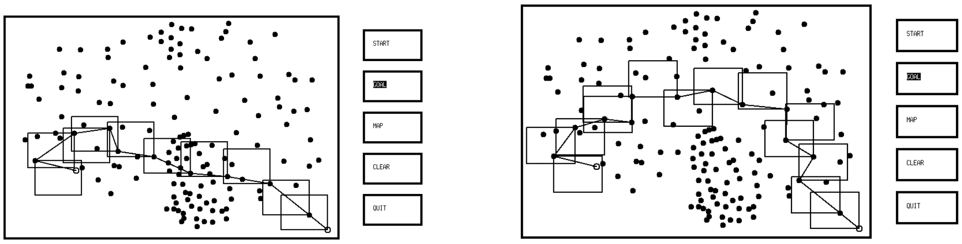

Figure 6: An example showing path that passes through a highly cluttered unreliable region (left), and the result of using the "reliability" criteria to avoid the cluttered regions (right)

man. *The Design and Analysis of Computer Algorithms.* Addison Wesley, 1974.

[KB88] Benjamin J. Kuipers and Yung-Tai Byun. A robust, qualitative approach to a spatial learning mobile robot. In *SPIE Cambridge Symposium on Optical and Optoelectronic Engineering, Advances in Intelligent Robotics Systems*, 1988.

[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*, 19(6):1656–1658, 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*, 2:129–153, 1978.

[LT90] Todd Levitt and Lawton Daryl T. Qualitative navigation for mobile robots. *Artificial Intelligence*, 44(3):305–360, 1990.

[Par93] Il-Pyung Park. *Qualitative Environmental Navigation: Theory and Practice.* PhD thesis, Columbia University, 1993.

[PK93] Il-Pyung Park and John R. Kender. Using isolated landmarks and trajectories in robot navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* IEEE Computer Society Press, 1993.

[ST93] Karen T. Sutherland and William B. Thompson. Inexact navigation. In *Proceedings of the IEEE Robotics and Automation*, 1993.

[ZT92] Jiang Yu Zheng and Saburo Tsuji. Panoramic representation for route recognition by a mobile robot. *International Journal of Computer Vision*, 9(1):55–76, October 1992.