ELSEVIER

Artificial Intelligence 78 (1995) 355-395

Artificial Intelligence

Topological direction-giving and visual navigation in large environments

Il-Pyung Park *, John R. Kender

Department of Computer Science, Columbia University, New York, NY 10027, USA

Received September 1993; revised January 1995

Abstract

In this paper, we propose and investigate a new model for robot navigation in large unstructured environments. Current models, which depend on metric information, have to deal with inherent mechanical and sensory errors. Instead we supply the navigator with qualitative information. Our model consists of two parts, a map-maker and a 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 landmarks that are continuously visible. Traversing on a trajectory enables the navigator to move reliably into featureless space, based on local headings formed by visible landmarks that are robust to positional and orientational errors. Reliability measures of parkway and trajectory traversals are defined by appropriate error models that account for the sensory errors of the navigator, the population of neighboring objects, and the rotational and translational errors of the navigator. The optimal path is further abstracted into a "custom map", which consists of a list of symbolic 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 the navigator are implemented using two cameras, an IBM 7575 robot arm, and a PIPE (Pipelined Image Processing Engine.)

* Corresponding author. Current address: Department of Computer Science, New York Institute of Technology, 1855 Broadway, New York, NY 10023, USA. E-mail: ip@faculty.nyit.edu.

SSDI 0004-3702(95)00030-5

1. Introduction

We are interested in navigation in a large and unstructured environment. An environment is regarded as large if at any instant the sensory and motor capacities of the navigating agent reside in a small subset of the whole terrain. Similar definitions can be found in [13]. An environment is unstructured if there is no way to use object identification to index into a global coordinate system. This may be because the objects themselves are easily confusable (like the trees in a forest), or because the navigating agent has no global referencing sensor (such as a compass), or because the navigating agent has no global map (so that even the availability of global coordinates is not useful).

Navigation in a large unstructured environment requires different information and tools than that of navigation in a small structured environment [13, 14, 24, 26]. The information that is useful is mostly qualitative rather than quantitative. High-level reasoning is necessary for the navigator to be able to handle ambiguities and errors; by high-level we mean symbolic direction-specifying invariants that can be used to capture the appropriate navigational information. For example, "continue in your present heading until you see an object on your right" is a high-level directional instruction, whereas "go true north for 200 meters and turn 40 degrees to the east" is not. One of the most important concepts in giving directions in a large environment is the use of landmarks. Without them, direction giving would be very hard for the navigator, as would be the verification that it has reached its destination. The topological relationships between landmarks are also useful, because these also tend to be insensitive to metric errors.

In this paper, we provide a new framework for topological navigation in a large unstructured environment. Our method emphasizes the use of a sequence of qualitative, high-level, landmark-based directional instructions that minimizes the navigator's efforts, and prevents, detects, and sometimes corrects navigational errors. 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,[1] (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 powerful direction giver (the map-maker) and a relatively simple direction follower (the navigator).

Our qualitative environmental navigation system, which consists of the map-maker and the navigator, can be categorized as both a preplanned and reactive type [22]. It is preplanned because the desired path of the navigator is computed by the map-maker subsystem. It is also reactive because the "custom map" provided by the map-maker for the navigator does not absolutely specify the desired path, but rather "describes" locally what the navigator's sensors are supposed to look for. Then, based on the activation of appropriate algorithms, the navigator reacts to the environment, usually progressing, but occasionally recovering from detected errors.

[1] This problem has been addressed by Kender et al. in [10], in which is presented the computational complexity of selecting an optimal set of sensors. Even in one-dimensional space, this proves to be an NP-complete problem.

2. Related work

The efficiency and the accuracy of navigation depends on the depth of the spatial knowledge of the navigating agent. Kuipers introduced the concept of "cognitive maps" to model the proficiency of the navigator, in which spatial knowledge consists of a hierarchy of "sensorimotor", "procedural", "topological", and "metrical" knowledge. Since the assimilation of knowledge proceeds from the lowest (sensorimotor) to the highest (metrical), it is generally more accurate for people to navigate by using lower-level knowledge, such as exact pictures of landmarks [12,27].

However, experiments done by Chase [4] indicate that novice drivers prefer topologically easier roads (major highways) over metrically shorter but more complicated roads (lesser known streets). Landmarks play an important role for these drivers; even with only a partial knowledge of the environment, they are able to navigate reliably by using visual cues. Streeter et al. [23] reported that people with low spatial abilities rely heavily on landmarks for navigation, and suggested that future route generating systems should produce a "customized" route that meets the individual's background skills. Cognitive map research, some of which have been described above, generally supports the importance of landmarks and their interconnection topology in navigation.

The navigating agent does not need to have a comprehensive knowledge of the environment for a single navigational task. For example, in order to get to a church for a wedding, the directions given by the host are usually enough. Experiments done by Streeter et al. [24] showed that verbal directions, which are roughly equivalent to Kuipers' procedural level in the cognitive map structure, are superior to the use of a more comprehensive global map. Riesbeck [21] asserted that the quality of directional instructions should be judged by the feasibility of the navigator to follow them, as opposed to how accurate they are in terms of the actual environmental geometry. Similarly, Mark's [17,18] experiments support the relative usefulness of procedural and topological information over metric information.

Traditional approaches to robot navigation require metric accuracy of the robot's paths. These methods include the configuration space by Lozano-Perez [15], generalized cones by Brooks [2], the segmented model by Crowley [5], the grid-based model by Moravec and Elfes [19], and the convex cell model by Giralt et al. [8]. Such traditional methods perform reasonably well only in small environments and fail in large unstructured environments [13]. Metric information becomes inaccurate, due to the low mechanical accuracy and sensory errors [26]; errors accumulate.

Qualitative approaches to robot navigation include: the TOUR model of Kuipers [12], the NX Robot by Kuipers et al. [13], Qualnav by Levitt et al. [14], inexact navigation by Sutherland et al. [26], Dai et al.'s "range-free navigation" [6], and the PV (Panoramic View) representation used by Zheng et al. [29].

Most of the work in qualitative navigation emphasizes the importance of landmarks. However, none provides a formal answer to the question "What is a landmark?", and they assume that landmarks can be readily identified by their intrinsic qualities. Otherwise, they use a generic definition of "distinctiveness" of features in the sensory readings to indicate landmarks. We argue that without criteria for defining and selecting good landmarks, topological navigation is not well formulated. In this paper, we present

qualitative methods to define and select landmarks, and we present means to navigate using these selected landmarks and their relative juxtapositions.

3. Definitions and assumptions

Our navigation system is composed of two modules: a powerful direction giver (the map-maker) and a relatively simple direction follower (the navigator). Navigational paths are precomputed by the map-maker and are composed of "parkways" and "trajectories". These are abstractions of the real world into topological representations. A parkway represents an interconnection network of landmarks, where the navigator can travel by following one continuously visible landmark to another. Paths between parkways are called trajectories, and they allow the navigator to move towards a currently invisible landmark based on the relative configuration of currently visible landmarks.

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 these instructions in a sequence. The vocabulary that constitutes custom map instructions is a set of qualitative descriptions of landmarks that can be easily and robustly processed by the navigator. The map-maker is assumed to be nearly omniscient and error-free. It sees the whole environment and knows the spatial coordinates of each object that exists. The map-maker also knows the sensor and processor limits of the navigator. Therefore, the custom map contains only 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 that the navigator's only source of directional information is the custom map. The custom map contains no absolute location or orientation information, and uses only small integers and few other symbols.

The navigator's capabilities are much more limited. Its view window size is very small relative to the environment, where the view window represents the extent of the navigator's vision relative to its current position.[2] It has limited metric measurement capabilities within its current view window, even though their accuracies are assumed to be low. For example, the navigator can "adjust" its physical position so that the currently visible landmark is located at approximately one of the four corners of the local view window (SE, SW, NE, NW). However, 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; even if it did, the navigator, having no compass, odometer, or other means of positioning itself with respect to an external global reference frame, would not be able to use it. The navigator is not intelligent enough to decide on its own what overall path it should select, thus it is totally dependent on the custom map that it is given. Therefore, the custom map can be considered as the navigator's intelligence. Table 1 summarizes the

[2] This limitation is similar to that of the bug robot with finite range vision in [16], where it is shown that, in some cases, increasing the range of vision of the navigator does not necessarily produce better results.

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

	Map-maker	Navigator
Objective	Generate a custom map	Navigate using a custom map
Visibility	Infinite	Limited to current view window
Metric ability	Yes; exact	Only within current view window; approximate
Intelligence	Nearly omniscient	Limited to interpreting the custom map
Memory	Large	Almost none
Computing power	Fast	Slow
Degrees of freedom	None needed	Two: x and y
Communication	Off-line	Off-line

assumptions that we make about the map-maker and the navigator, some of which are further explained below.

3.1. *The world, the map-maker, and the navigator*

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

- *The real world* This is 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 relatively small objects. Therefore, we need to abstract this world into something simpler.
- *The world model of the map-maker* The first level of abstraction is done by the map-maker, who is nearly 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 regions without intervening objects that are navigable under certain constraints: this graph is the topology. Empty space, distance, and object extent are generally ignored. (However, these can be, and are, considered by the map-maker when computing "optimal" directions and paths.) The data structures used in this level of abstraction are parkways and trajectories. The formal definitions of parkways and trajectories will be given in the next section.
- *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 of visual landmarks, and accompanying actions, ordered by the sequence of traversal. This custom map contains the minimal 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 near-omniscience extends to a perfect model of the navigator's sensory endowment and error behavior. The navigator perceives the world in an extremely limited way. Sensory range is limited to a relatively small area of the world (the view window), and there is almost no memory. However, this is actually an advantage; usually, the less that needs to be communicated, sensed, or remembered the better, since the cognitive load of the navigator is minimized [24]. Although more intelligent navigators can be modeled, this one chooses to

ignore most of the world and its features for the sake of efficient map-making. The custom map[3] in its most simple case is represented in the form: $(D, M)^*$, where D is a qualitative description which distinguishes a landmark from its neighboring objects, and M is the movement of the robot in the local robot coordinate system established by using the chosen landmark as the reference point. This sequence of descriptions and relative movements can be encoded very compactly into a short string of symbols.

Our navigator robot has two degrees of freedom, in x and y. It is able to view a small subarea of the environment, looking straight down, perpendicular to the environment; the z position of the navigator is fixed. Thus we are in a "level-helicopter world" as in [11]. For our experiments, we implemented the navigator with an IBM 7575 SCARA robot arm with a CCD camera attached to the robot's end effector. The navigation terrain was the robot arm's workspace.

3.2. *The navigable world*

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 whose background is visually uniform all over. Objects are scattered over this flat surface; they are spherical, such as marbles, and uniform in size. There is no restriction on where the objects are placed, except that no objects are allowed to be placed on top of another. These 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". That is, an object can no longer be described by its intrinsic attributes, such as shape, size, or color, since they are all identical. And in order to specify an object, the geometrical relationships of the object to its neighboring objects must be considered since there is no apparent global reference frame. Further, there is no "natural" environmental structure to rely on for the navigation. In other words, this is equivalent to doing navigation without an odometer, a compass, or defined roadways. This abstraction is therefore extreme, but intentionally so—it is designed to explore what can be done using very impoverished sensing, identification, and movement. Most other navigational environments will therefore be easier, but a fortiori, they too can exploit the results of this model, particularly in their more difficult areas.

In a more complicated world that has a variety of different objects, we can (and often humans do) still apply the same abstraction. For example, in the left diagram of Fig. 1, we see a distribution of houses and trees. In the simplest form of abstraction, every house or tree can be regarded as a point-like object, and the only information extracted by the map-maker is its surface location in the environment. This is shown on the right diagram of Fig. 1. But another way of abstracting the world would be to use sensing to disambiguate the objects according to their types, using whatever categorizational

[3] Later in this paper, we present a formal grammar of the custom map that includes trajectories and error recovery instructions.

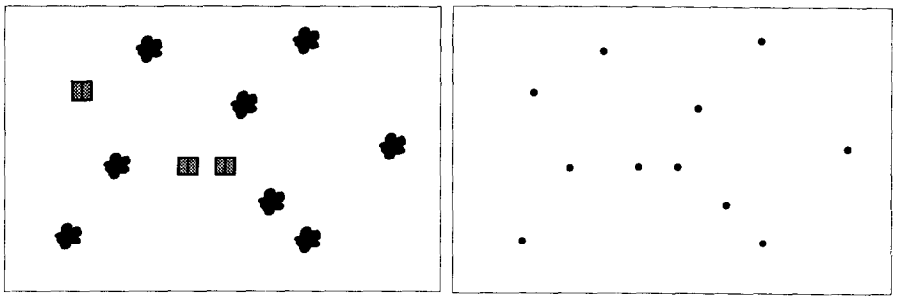


Fig. 1. A world consisting of trees and houses (left), and the corresponding dots world representation (right).

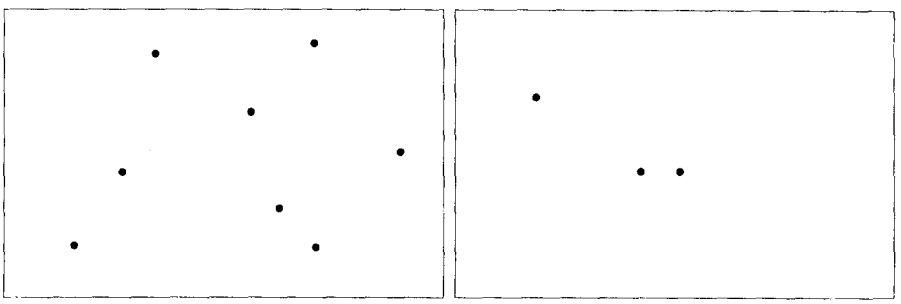


Fig. 2. A separation of the dots world according the navigator's sensibility: only trees (left), or only houses (right).

strategy is available to the navigator—this produces a number of separate such "dots worlds". This is shown in Fig. 2. So, if there are features that are sensible by the navigator, they are used to filter the perceptions of the map-maker and the navigator. Then, each world becomes simpler, and relative relations become more reliably apparent since it consists only of point-like objects of the corresponding type. In short, being able to distinguish objects makes a world easier to get around in, but topological problems remain; the model we present addresses these problems under the extreme abstraction that any object is considered a potential landmark for navigation.

4. Parkways and trajectories

In this section, we describe two fundamental data structures, created and used by the map-maker for representing the interconnection topology of landmarks, called parkways and trajectories. They are graphs representing the topological information of the environment as well as some associated costs, such as the reliability of traversal and the expected traversal time. Parkways generally provide a major component of a custom map path, enabling the navigator to travel by following continuously visible landmarks. Trajectories provide the navigator with a way to navigate the featureless space between parkways; no landmarks are visible for most of a trajectory.

4.1. Parkway definition

When the map-maker captures the real world with its global sensor, it records all the positions of the populating objects. As stated earlier, we assume that objects are identical in size and shape. Therefore, the intrinsic quality of each object is not useful for navigation; only the (relative) location is. The map-maker further abstracts this world into a graph data structure of vertices and edges. Each vertex represents an object and each edge indicates a navigable 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 to be connected if the two objects can be viewed in the same single view window. 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; the navigator can then move towards the goal, while keeping the goal continuously visible.

By applying a connected component algorithm [1] 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 landmark must be (visually) reachable from the previous one. The left diagram of Fig. 3 shows an example of the map-maker's view of a randomly populated world of size 100×100 with 100 objects. The right diagram shows the parkways formed on the corresponding world for a navigator whose window size is 10×10. The relative size of the view window is shown in the lower left corner of the world. The arcs between objects indicate that the objects are connected. That is, two objects are connected only if there is a way to place the 10×10 window so that both objects are simultaneously visible in it.

The regions in Fig. 3 with various shades indicate the spatial reach of each parkway. When a new object is added to the environment, it is absorbed into the parkway corresponding to the shade, if such a shading exists. That is, the shaded regions denote where a new object can be seen simultaneously by some current element of an existing parkway; unshaded regions indicate where new objects would start new parkways.

To find the best path between two objects in the same parkway, we apply Dijkstra's shortest path algorithm to the parkway graph, which also has recorded certain costs (which will be detailed later). Having selected an object as the source node, the graph is explored and the costs are accumulated, in the usual way specified by the algorithm. The significance in this model, however, is that a resulting path is a particularly reliable one for the navigator. Starting at the source, it can position its window so that the designated next node becomes visible; by definition, this next node can be seen at the same time as the source node. The navigator then can proceed away from the source to this next node, without the loss of visibility of this next node. Errors, therefore, usually cannot accumulate; the navigator seeks only what is already visible. As the navigator continues, each successive landmark is identified and "sought", that is, each successive landmark causes forward progress; then the sought landmark itself becomes a reference landmark for the next step. Therefore, navigation within a parkway is relatively "easy". Exactly how easy it is to navigate within a parkway will depend on the complexity and

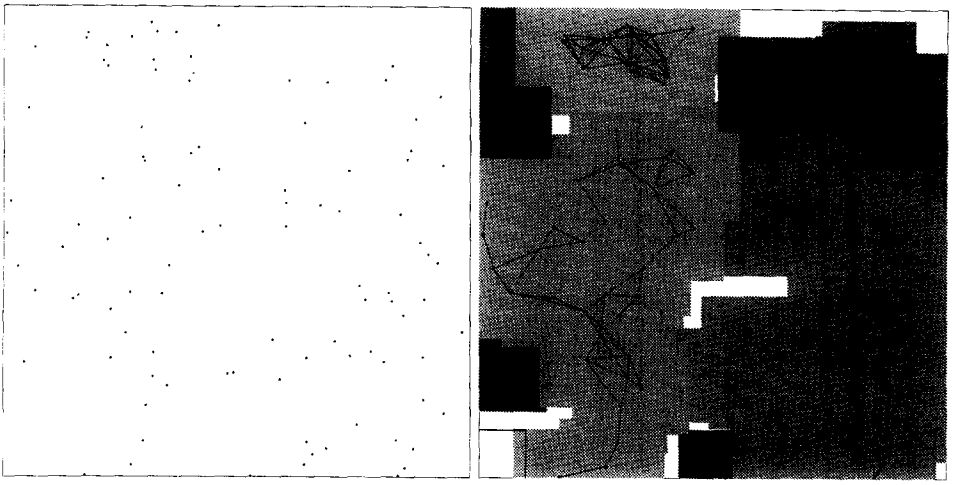


Fig. 3. A randomly populated dots world (left) and the corresponding parkways (right).

the reliability of the description language for identifying landmarks, which we define in Section 5.

4.2. Types of parkways

In the previous subsection, we defined the parkway in the most general sense. However, depending on the definition of connectedness, there can be many different parkway types. In this subsection, we introduce two useful subclasses, the "obvious" and the "isolated" parkways. Other parkways are also possible, by using different descriptors (see Section 5).

4.2.1. Obvious parkways

If the number of objects within the view window is more than two, the map-maker needs a qualitative way to describe to the navigator which of the objects is intended to be sought next by the navigator. This is a hard problem, and will be discussed in more detail in Section 5. However, in the simplest case, there are only two objects in the view window. (The case of just one object represents a "dead end".)

We define an object to be an "obvious landmark" if it is the only object visible, other than the current reference landmark. For simplicity in the implementation and for utility in making the largest amount of the world visible, we adopt the convention that whenever the navigator seeks a landmark, it does so by maneuvering itself so that the landmark becomes visible in one of the four corners (SE, SW, NE, or NW) of the view window.[4] Thus each reference landmark in the environment can have up to four obvious

[4] Other definitions of connectedness are possible, depending on sensor limitations. For example, in the case of a helicopter with an oblique view of the world, the connection can be that the navigator positions itself so that the reference landmark is in the lower middle of the view window: the standard "windshield view" of the world.

Fig. 4. The left diagram shows the original world with obvious landmarks illustrated in bigger circles. The right diagram shows corresponding obvious parkways. In the right diagram, the box in the lower left corner represents the relative size of the navigator's view window.

landmarks, one for each corner position of the reference landmark. The parkway formed by connecting obvious landmarks is called an "obvious landmark parkway", and for short we will refer to it as an "obvious parkway". Note that unlike a general parkway, an obvious parkway is a directed graph and therefore the paths are not commutative: what is obvious from one view point may not be obvious in reverse. Fig. 4 shows the abstraction of the original dots world of Fig. 3 into obvious parkways; although not apparent in the arcs in the figure, the graph is directed.

4.2.2. Isolated parkways

A second special case of parkways is based on the concepts of landmark isolation. As will be shown, isolatedness is a robust topological property relatively invariant to sensor error, so it is well-suited to qualitative navigation. Given a scene with one or more objects, humans often can identify "the most isolated" object. A detailed algorithm to derive the isolated landmark in a scene based on the *mutual neighborhood* concept will be given in Section 5.

We define the parkway based on landmark isolation in the following way. As in the obvious parkway case, we adopt the convention that the navigator seeks a landmark by positioning itself so that the landmark becomes visible in one of the corners of its view window. This landmark is now the reference landmark. Any such landmark now may have up to one isolated landmark per corner position. A parkway formed by such isolated landmarks is called an "isolated landmark parkway", and for short we will call it an "isolated parkway". Fig. 5 shows the abstraction of the original dots world into isolated parkways. Note that an isolated parkway is also a directed graph, and that it is a superset of the obvious parkway, but a subset of the general parkway.

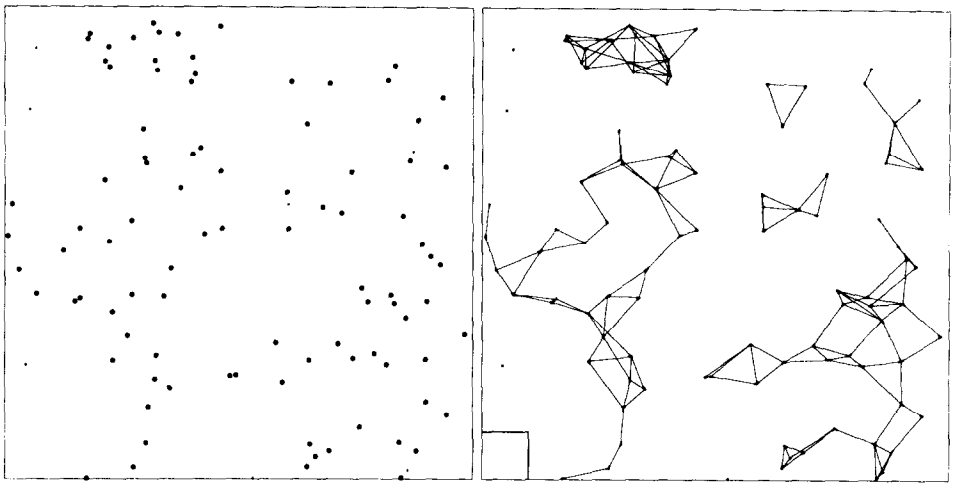


Fig. 5. The left diagram shows the original world with isolated landmarks illustrated in bigger circles. The right diagram shows corresponding isolated parkways.

4.2.3. *Reliable parkways*

The obvious parkway and the isolated parkway turn out to be two of the most useful concepts for reliable navigation. However, other types of parkways are possible. For example, each parkway path can be associated with a reliability measure, similar to a probability, as will be explained in Section 6. Parkway paths can then be categorized by whether or not they exceed a reliability value threshold. This defines a definition of connectedness that results in "reliable" parkways. Note that reliable parkways are neither a strict superset nor a strict subset of isolated parkways.

4.2.4. *Error-detectable and error-recoverable parkways*

During navigation, errors and mistakes are sometimes inevitable. In some errorful situations, however, it is possible for the navigator to identify that it is no longer on the correct course. Furthermore, in some more restricted cases, the navigator can recover from an error by executing a simple recovery sequence. This process of identifying and recovering from an error is of a great importance [25], and thus will be explored in detail in Section 6. This concept leads to another classification of parkways.

A subgraph of a parkway that is entirely composed of error-detectable directed edges is called an error-detectable parkway. Likewise, if the edges represent error-recoverable paths, then the parkway is called an error-recoverable parkway. For example, a subgraph of an isolated parkway that is error-detectable is an error-detectable isolated parkway. Again, these properties of parkways are not easily related to each other. Reliability, error detection, and error correction are defined by performance criteria, whereas obvious, isolated, and (general) parkways are defined by visual connectedness.

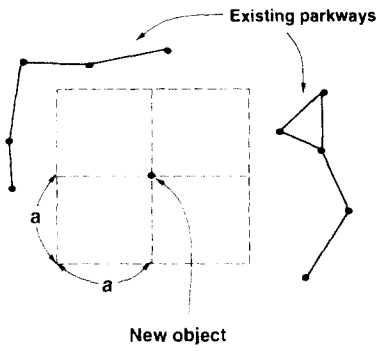


Fig. 6. A new object is added, and forms a new parkway, since it is not visible to any other existing parkway. The window size is $(a \times a)$.

4.3. Statistical analysis of parkways

We have carried out some experiments to determine the relationships between object density, the window size, and the number of parkways of various types. Let the world size be $R = (r \times r)$, and the window size be $A = (a \times a)$. We proceed, via simulation, as follows. At each integer time interval, we add (using a uniform random distribution) a single new object to the world, and see if it starts a new parkway or is incorporated into an existing parkway. The number of objects at time t is therefore t, the density of objects in the world is t/R, and the average population in a window is $(A/R)t$. Since we are assuming a random placement of the objects, the number of objects in a window can be approximated by a Poisson random variable.

The probability of forming a new parkway at time t can now be calculated. We place the tth object into the world and observe its surrounding area. If there are no other objects within one window's distance of the new object, then a new parkway is formed (Fig. 6). Therefore, the probability of forming a new parkway at time t is $e^{-(4A/R)t}$ $(= (e^{-\lambda}\lambda^0/0!)^4$, where the Poisson parameter, $\lambda = (A/R)t$, represents the average population in the window).

In the top diagram of Fig. 7, the results of a simulation of this concept are shown. The number of parkways formed versus the population of objects is plotted. The five line graphs from top to bottom are associated with window sizes of 2×2, 3×3, 4×4, 5×5, and 10×10, respectively. In general, the size[5] of an average general parkway in an environment is larger than that of an average isolated parkway or an obvious parkway. Conversely, the number of disjoint general parkways formed in an environment is less than that of isolated parkways or obvious parkways. The bottom diagram of Fig. 7 shows a comparison of the three parkway types when the window size is 5×5. Intuitively, there is a time t corresponding to a density of landmarks that maximizes the number of parkways of a given type. Above this density, parkways coalesce. We speculate that "difficult environments" for navigation are those near this

[5] The size of a parkway can be defined in many ways, but in general, the number of nodes in a parkway is a good measure.

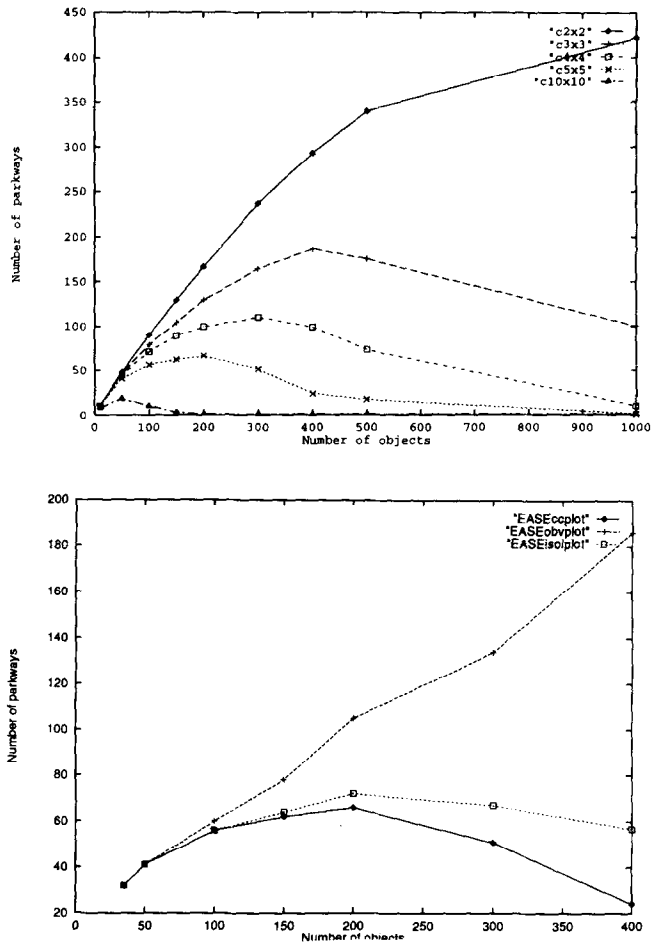


Fig. 7. Top diagram: The number of parkways versus the population. Bottom diagram: Number of obvious, isolated, and general (in top to bottom order) parkways versus the population: the more restrictive the definition of connectedness, the more likely the parkways will be smaller in size, and therefore more numerous.

relative maximum of parkways, since at this density the largest number of landmarks are not in immediate visual range of each other.

4.4. Trajectory definition

Sometimes, the start and the goal landmarks can be in mutually disjoint parkways. In this case, following a parkway will not be sufficient for the navigator to accomplish its task of getting to the goal. Therefore, we need a method to transit between two parkways. At some point of the traversal, the navigator has to leave a parkway, enter a region of featureless space, and attain the other parkway without getting lost. Recall from our original assumptions that the navigator does not have any metric capabilities

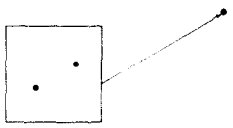


Fig. 8. An example showing how a trajectory is formed.

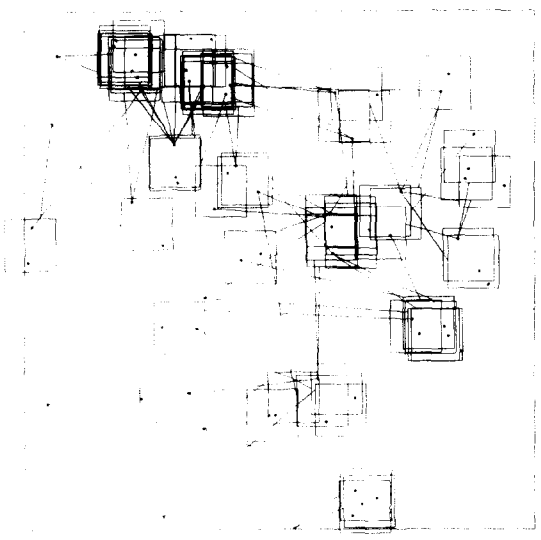


Fig. 9. Trajectories formed on the environment shown in Fig. 3.

outside its view window. Thus, we need a method that utilizes the relative locations formed by currently visible landmarks to determine a heading that the navigator can follow to another parkway. In order for the method to be robust, this method needs to be as insensitive as possible to rotational, translational, and scale errors.

We propose the following. For each pair of landmarks that can be seen in the same view window, the map-maker tests the feasibility of reaching a landmark in some other parkway by "sliding" the window in the direction formed by the two landmarks. This is depicted in Fig. 8, where the two objects in the view window form a directional vector, and the third object in the upper right corner can be brought into the view window by sliding the view window in this direction. The trajectory of the window movement is represented by the straight line.

Inter-parkway paths generated by this method are defined as "trajectories". It is interesting to note that trajectories are not commutative, and the graph formed by trajectories is in fact a directed graph. One measure of the ability of trajectories to add connectedness to disjoint parkways is to apply the strongly connected component algorithm to this directed graph [1]. Fig. 9 shows an example of trajectories computed on our random world. Each straight line is the trajectory of the window, in the direction of the sliding movement. One end of a line is attached to the window at the position at which the new object is expected to appear. The other end of the line is the object that this sliding window is seeking. Note that for much of the trajectory, the navigator sees

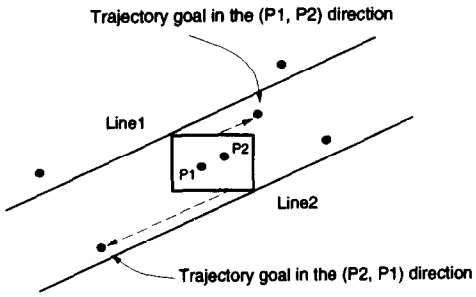


Fig. 10. Formulation of trajectories.

nothing at all, and that the attaining of the new landmark occurs along the leading half of the perimeter of this window.

4.5. Implementation of trajectory traversal

The basic idea in the trajectory approach is to "slide" the view window along the direction formed by two landmarks within the view window and to see which landmark enters the window first. Note that the map-maker does not use any global coordinate system to specify the navigator's direction of movement. In Fig. 10, we see the problem from the viewpoint of the map-maker: a view window is centered around two objects, P_1 and P_2. Although the map-maker can compute the global coordinates, (x_1, y_1) and (x_2, y_2) of these points, from the viewpoint of the navigator, it is only the local coordinates in the navigator's window coordinate system that will be available to the navigator. However, in either view, the direction formed by these two objects, P_1 and P_2, is defined by the line that passes through them. The "goals" are then the closest objects to this window along the formed directions[6] (left or right).

4.6. Types of trajectories

In order to form a trajectory, the navigator uses two visible landmarks. Since the navigator has a reference object at all times, it needs to identify a single landmark to make this pair. Depending on the description language, different objects can be selected as the second landmark, which would yield different trajectories. For example, using the isolated landmark descriptor, a trajectory can be defined by the reference landmark and the isolated landmark. A trajectory formed by using the isolated landmark descriptor is called an isolated trajectory. Likewise, a trajectory formed by using the obvious landmark descriptor is called an obvious trajectory.

As in the parkway case, each trajectory path can be associated with a reliability measure, as explained in Section 6. Trajectory paths can then be categorized by reliability. In Section 6, we also describe methods for error detection and error recovery during

[6] The specific formulation of a trajectory as described above applies to a rectangularly shaped view window. However, the basic idea of trajectories can be generalized to an arbitrarily shaped window, or to navigation in dimensions higher than two.

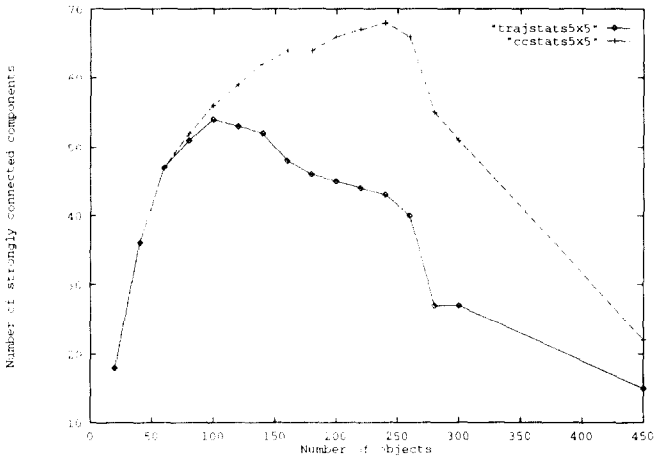


Fig. 11. Number of strongly connected components as a result of applying the trajectory method.

trajectory movement. Error-detectable trajectory paths and error-recoverable trajectory paths are defined in analogous ways to their parkway counterparts.

4.7. Statistical analysis of trajectories

As a result of forming trajectory paths, the number of (strongly) connected components decreases. This means the world is more connected as a whole and easier to navigate in. If there are no trajectory paths, the number of strongly connected components is equal to the number of existing parkways. In the simulation, the upper curve of Fig. 11 represents the number of parkways as the number of objects increase. The lower curve indicates the number of strongly connected components. Note that trajectories are "one-way", and that the measure of connectedness displayed in the graph is strong connectedness. That is, the parkways must be connected by two separate trajectories so that navigation is possible in both directions. The graph therefore indicates a lower bound (i.e., you can usually do better) on whether or not you can get from one place to another. Note that even at a relatively high population, for example at 450, the number of strongly connected components is not 1. This means that even by using both parkways and trajectories, there are still places that "you can't get to". The major advantage of trajectories is that they make the world "smaller" (roughly by a factor of two) after a certain density.

4.8. Overall best path

In general, the overall best path is a combination of parkway path segments and trajectory path segments. An example of such a path is shown in Fig. 12. The starting position is near the top right corner, and the goal position is near the bottom left corner. Parkway path segments are represented by single line segments, and the trajectory path segments are represented by a box accompanied by a single line segment. Without much

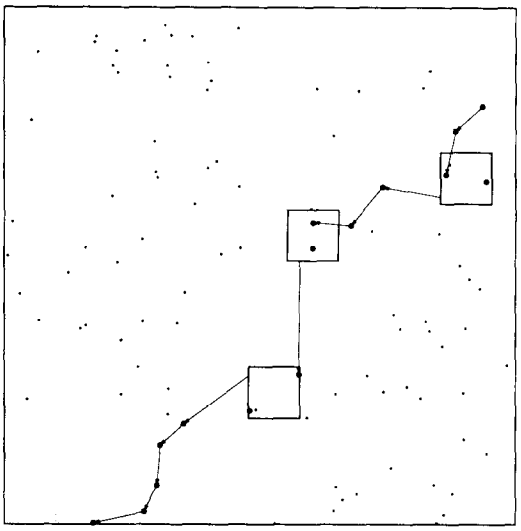


Fig. 12. The combined shortest path.

change to Dijkstra's algorithm, the best path can be easily generated, incorporating both types of navigation.

5. Path and custom map generation

The main objectives of the map-maker are to compute the navigational path, and to generate a corresponding custom map of symbolic topological directions for the navigator. Only then is the navigator able to start on its journey. The map-maker must decide which route is the "best" for the navigator to take. But, usually, the shortest distance route is not the best route to take when the driver is not familiar with the environment. The map-maker must take into consideration various costs for the navigation, some of which are: how far is the travel distance, how much time does it take, how "easy" is the path to follow [4], how "easy" is the path to describe [17], and how to provide enough information without overburdening the navigator [24]. The first three items in the list are related to the derivation of paths. Depending on the skills and the experience of the navigator, the priority and relative weights of these costs may vary. Also, depending on the priority of each of these costs, the resulting path may vary. For example, the shortest distance path may not be the shortest time travel path. Furthermore, the easiest (that is, the most reliable) path may be drastically different from those two. If the navigator is unskilled or is not familiar with the environment, the path derivation should favor easy paths over faster paths. The latter two items in the list are related to the direction giving. Describing how to follow an already derived path is as important as the path generation itself.

Traditionally, robot navigation theory has been divided into two groups. The first is the traditional AI theory applied to navigation, where high-level planning is done prior

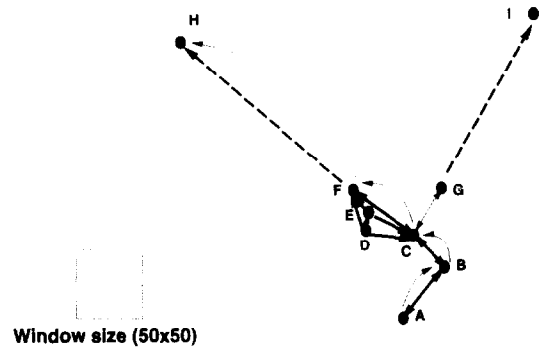


Fig. 13. Example of isolated parkway paths and trajectory paths.

to navigation. The other is the reactive control theory, such as Brook's subsumption architecture [3], where navigation is low-level: the robot navigates by reacting to the environment based on its sensory inputs. Our map-maker and navigator models incorporate both of these two approaches. The map-maker's path planning and custom map generation provides a high-level preplanned control of the navigator. The navigator's actions are based on the custom map commands and the immediate environmental sensory reading, which can be regarded as the reactive control of the robot navigator.

The map-maker uses its knowledge of the environment (represented in data structures abstracted in topological forms) and of the navigator's characteristics (view window size, degree of freedom, accuracy, etc.) to produce a relatively short and simple direction sequence (the custom map) for the navigator's use. The directions in a custom map have a simple grammar that describes the next landmark to be sought, and that describes how to navigate and adjust the local position while the reference landmark is visible. All these aspects of the custom map construction and execution are now described in detail.

5.1. *Path generation*

The transition matrix contains information as to whether or not a landmark can be navigated to, from another landmark. That is, if object B is directly attainable by the navigator from object A without going through another landmark, the entry of the transition matrix corresponding to (A, B) is set to a value representing the cost of this navigation.

Let us refer to Fig. 13. The solid straight lines represent isolated parkway path segments, and the dotted straight lines represent trajectory path segments. The square box shown to the left of the figure is the window size used to calculate the parkways. (Recall that local visibility in the navigator's window is the key factor in determining the connectedness.) The number of disjoint isolated parkways is four because there are four strongly connected components. Objects A, B, C, F and G are in the first isolated parkway. Objects D and E are in the second isolated parkway; this is because the transition from object group D, E to A, B, C, F, G is one way. Objects H and I are each on its own parkway.

Table 2
Transition matrix

	a	b	c	d	e	f	g	h	i
a		40.31							
b	40.31		38.60						
c		38.60				52.35	59.55	132.52	159.93
d			34.13		16.76	35.06			
e			34.67	16.76		20.25		132.52	
f			52.35						
g			59.55						159.93
h									
i									

From the observation of the transition matrix shown in Table 2, we immediately see that object C is a crucial landmark in this environment because its corresponding column is the most filled. Thus, an operational definition of a "good" landmark emerges: it is an object that is reachable by the navigator from many other objects. Similarly, the "best" landmark is one that is reachable from the most objects. It is important to notice that the quality of a landmark depends on the sensory limitations of the navigator (here, the view window), as well as the qualitative description method (here, the navigator's ability to detect isolatedness). Nevertheless, detectability and reachability by the navigator appear to be the two defining characteristics of an object useful in topological direction giving and following: these are what make and quantify a landmark.

Each entry in the transition matrix is a value indicating the cost of the transition. Depending on the need of the navigator, this value can bear different meanings. For example, this can be the spatial distance, the temporal distance, the reliability, or any other weighted index of evaluation that can be measured or generated.

The shortest path in a parkway is generated by Dijkstra's algorithm; it finds the single-source shortest path to all other vertices. This requires $O(n^2)$ time, where n is the number of vertices in the graph. The overall shortest path is a combination of parkway paths and trajectory paths. In our example of Fig. 13, the shortest distance path from object A to object H is represented by a sequence of arrowed arcs: three parkway transitions, followed by one trajectory transition.

5.2. *Tie breaking heuristics*

Often Dijkstra's algorithm will have the option of selecting more than one landmark at equal cost. These ties are broken by means of heuristics.

Sometimes, during an isolated parkway traversal, two or more objects in the navigator's view window will seem equally isolated to the navigator, i.e., they have the same c-values, described later in this section. To break the tie, the map-maker uses a heuristic to choose the isolated landmark that is furthest away from the current reference landmark. This method is based on the observation that the navigator tends to travel towards the goal (and away from the starting location) at any instance of navigation. Therefore, by selecting the furthest landmark from the current reference landmark, the navigator usually moves closer to the goal. Note that this is in contrast to a more usual search

heuristic, which would select the landmark closest to the goal. The reason we use the former is that the global position of the goal (or even of the two competing landmarks) is unknown to the topologically driven navigator. The problem is subtle: even though the map-maker does know global positions, the navigator does not. Hence, any heuristic used by the map-maker in planning must also be usable during the "reactive" sensing by the navigator as well.

For trajectories, the orientational (directional) error in following a trajectory is minimized when the intercepting edge of the view window and the direction of the trajectory are at right-angles. Further, the error in determining a trajectory direction is inversely proportional to the distance between the two reference objects that define the direction of travel [20]. Therefore, the heuristic of choosing the object that is furthest away from the current reference landmark also applies to the trajectory task, since it tends to minimize error. Here, too, the capabilities of the navigator are respected: if there is a tie within a view window, the furthest landmark can and will be selected by the navigator as well.

5.3. *Definition of optimal path*

Depending on the needs of the navigator, different constraints can be imposed into the computation of the optimal path. Some examples of optimality criteria are: the travel distance (D), the reliability (R), the travel time (T), the sensing cost (S), and the custom map length (M).

Currently, we have implemented cost functions that estimate the travel distance of a navigation path, D, and the reliability of a navigation path, R. (Often, T is minimized when D is; and in an earlier separate work [11] we have implemented S and M.) One of the map-maker's responsibilities 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 with each other. For example, if the shortest distance path involves a highly cluttered area, the reliability would be low. Conversely, a reliable path that avoids highly cluttered areas may force the navigator to detour around a shortest path. There are many ways to weigh these two costs. Here, we suggest a third function, C, that merges D and R, defined as D/R: for simplicity of computation it can be rewritten 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

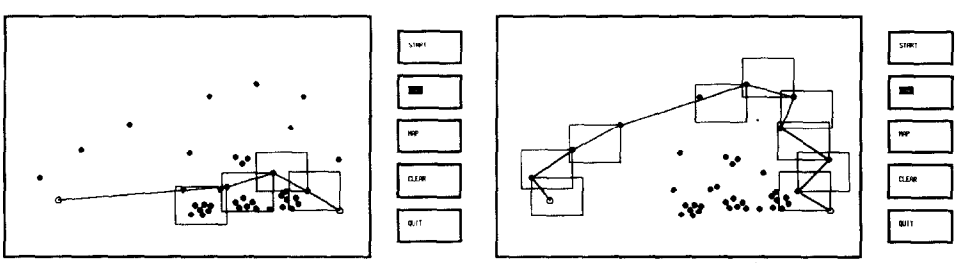


Fig. 14. The shortest distance (left) and the most reliable (right) paths.

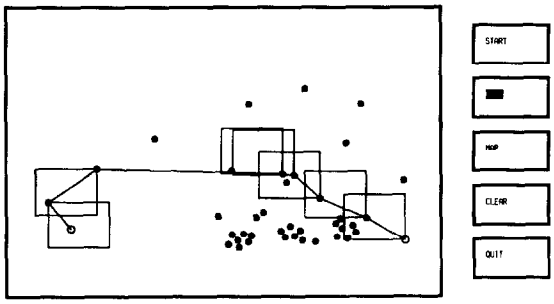


Fig. 15. The D/R compromised path.

path will tend to favor short paths and sparsely populated areas. Note that R is a function of not only n (the population of landmarks) but also σ (the position estimate error) as will be shown in Eq. (1) in Section 6. 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. Examples of the shortest distance path, the most reliable path, and the D/R compromised path are shown in Figs. 14 and 15.

5.4. *Effect of sensory context on landmarks*

It is not surprising that the shortest path in a parkway network does not necessarily guarantee the shortest travel path for the navigator. This is because the cost (distance traveled) of achieving a landmark depends on what might be called the current sensory context of the landmark. For example, consider a path generated within a parkway. Each arc of the path represents the distance between a reference landmark and a subsequent landmark. However, depending on which corner of the view window (SE, SW, NE, NW) the new landmark is to be placed, the actual traveled distance of the navigator may or may not be equal to the arc length. In an extreme situation, we can visualize a parkway path that zigzags, but in which the actual traveling movement of the navigator is nearly linear and the traveled distance is much smaller (see Fig. 16). Conversely, a parkway path can be nearly straight, but the sensing demands of the navigator may force a more jagged movement of the navigator.

Using our navigator model, each target landmark has four different sensory contexts, corresponding to the reference landmark being placed in SE, SW, NE, and NW corners of the navigator's view window. In order to implement context-based landmark following, the data structures that represent parkways, trajectories, and the transition matrix, must be modified. Instead of n nodes in the parkway network, we now have $4n$ nodes, namely, the four ways each target landmark can be "seen" by the navigator. The size of the transition matrix grows by a factor of 4^2, but stays very sparse. Therefore, a sparse matrix representation can be used for storage efficiency. The time complexity of the search algorithm increases, also by a factor of 4^2 since Dijkstra takes $O(n^2)$.

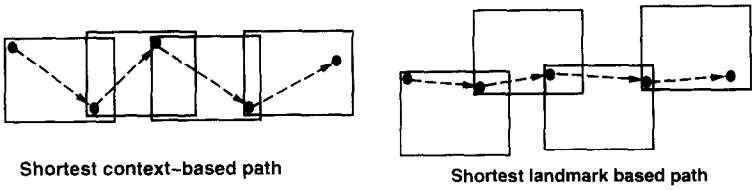


Fig. 16. The arrows indicate physical distances between landmarks, whereas the actual movements are shown by the rectangular boxes.

5.5. Custom map generation

For the actual navigational task, the navigator depends solely on the custom map created by the navigator. The custom map contains the navigational instructions that correspond to parkway or trajectory path segments. Each entry in the custom map instructs the navigator (1) what to look for (the "seek" phase), (2) how to verify the landmark (error detection), (3) how to correct itself (error recovery), and (4) where to go with respect to the landmark (the "adjust" phase). Later in this subsection, we will discuss the environmental description language for qualitatively identifying a landmark. The vocabulary in this description language defines the command set available to the navigator. However, independent of the vocabulary, the grammar of the custom map is particularly simple, and we cover it first.

Each entry in the custom map is of the form $(D, [T,][E,]M)$. D is a description language vocabulary symbol that indicates which landmark in the current view is to be "sought". Most often, this will call for the most isolated landmark. $[T]$ is an optional term that indicates a trajectory, where $T \in \{\text{Trajp, Trajn}\}$. Trajp (Trajn) indicates a trajectory in the positive (negative) direction from the reference landmark to the isolated landmark. $[E]$ is another optional term which consists of a set of instructions for error recovery sequences that will be described in Section 6. Finally, M is the corner designator to indicate which of the four corners (SE, SW, NE, NW) the chosen landmark is to be "adjusted" to once it is attained. Symbols $*$ and $-$ are used to indicate "any" and "null", respectively. For example, the direction (*, SW) means to seek any object in the view window and to adjust until it is in the SW corner. The navigator in following this direction selects a landmark, and then moves so that the selected landmark appears in the designated corner of the view window. (Surprisingly, in some environments, "any" is a powerful description; these are environments in which "you can't miss it".) The direction (Obvious, –) means that the navigator can select the obvious landmark and do nothing. (Usually, this means that it has reached the goal, or, in error correction, that the correct landmark has finally been attained.)

Some examples are shown in Fig. 17. In the simplest case, we have the custom map entry, (Isol, SW) in upper left diagram of Fig. 17, in which the navigator is told to identify the isolated landmark and then to adjust to the SW corner. The upper right diagram shows the custom map entry, (Isol, Trajp, SE), in which the navigator is told to identify the isolated landmark. Then, by moving in the direction formed by the two landmarks (the reference landmark and the isolated landmark), the trajectory

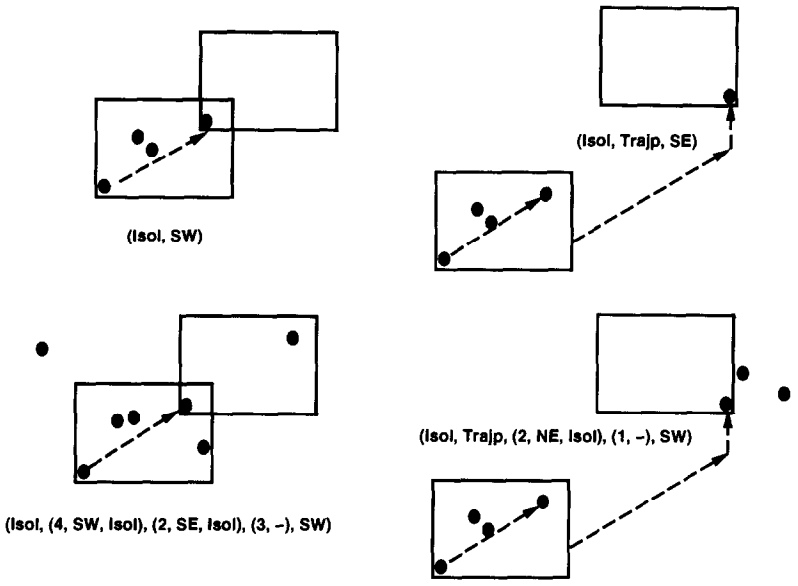


Fig. 17. Examples of custom map entries.

goal landmark is brought into the view window, and the navigator adjusts to the SE corner. The custom map entry becomes more complex when error recovery instructions are included, as in the lower diagrams of Fig. 17, as will be explained later in Section 6.

5.6. *Details of the 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. In this subsection, 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 in its view, which may contain several objects. The map-maker 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 landmark. This is a hard problem because of our assumption of the navigable environment, which is comprised of point-like objects that are randomly placed. (Again, this abstraction has been deliberately chosen to stress topological relationships. If objects have distinguishable features such as color or shape, those can be used in the description. Or, if they share certain detectable spatial regularities, such as being placed in a square-like configuration, these properties can also be used in the description. What follows is the extreme case where no features are available, or, equivalently, when all objects have the same features.) If the navigator had infinitesimal accuracy, infinite memory, and an extremely fast processor, the description language could consist of a full color image of each possible view. But since our world is a monochrome point-like world anyway, we need a qualitative language that can describe the geometric and the topological relations of the visible objects. The level of detail in the description process would depend on the

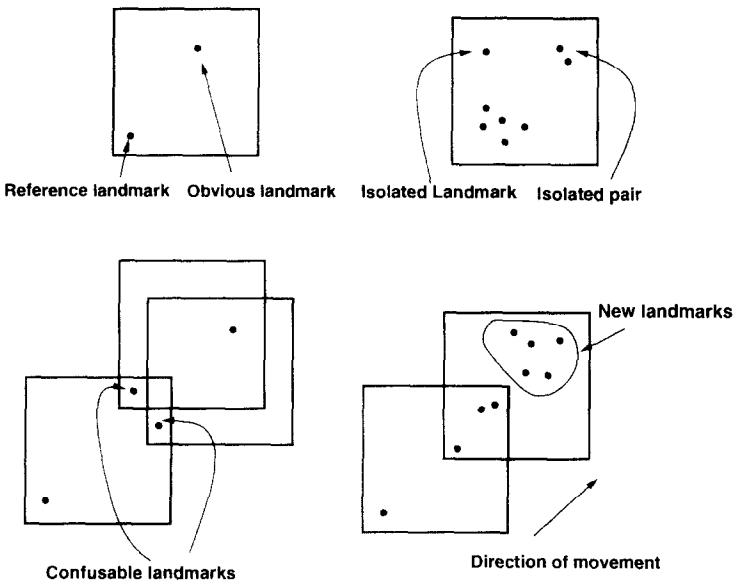


Fig. 18. Examples of the description language, only two of which (obvious and isolated landmarks) are fully analyzed and implemented in this paper.

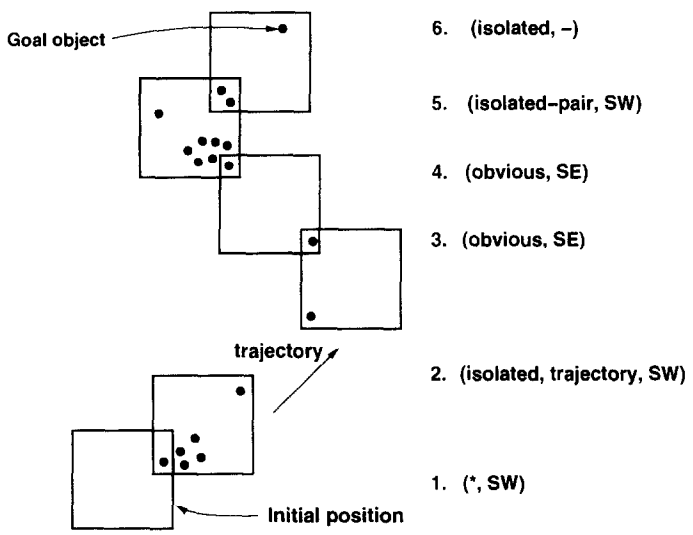
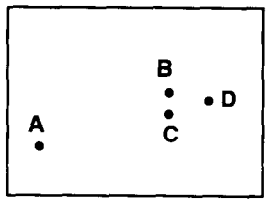


Fig. 19. An example of navigation using a custom map, showing isolated landmarks, obvious landmarks, a trajectory, and an isolated pair. Corresponding custom map entries are shown on the right.

intelligence and the sensory ability of the navigator; our exploration here deliberately emphasizes the topological aspects of navigation.

Some definitions follow: A landmark is defined to be "obvious" if it is the only landmark that is visible except for the reference landmark. Two or more landmarks are



	A	B	C	D
A	0	5	4	6
B	5	0	2	3
C	4	2	0	4
D	6	3	4	0

Fig. 20. An isolated landmark example (left) and the corresponding *mnv* matrix (right).

defined to be "confusable" if it does not matter which landmark is to be chosen as the next reference landmark (again, some environments do permit this). The term "new" landmarks refers to the landmarks that newly came into the view window as a result of the robot's movement. The term "isolated" landmark refers to a single isolated landmark as a result of applying a clustering algorithm. The term "isolated pair" refers to a single pair of isolated landmarks as a result of a clustering algorithm. Fig. 18 illustrates these definitions. All have been or are being implemented, but we will only talk about obvious and isolated landmarks in this paper. This is because isolated landmarks were surprisingly robust and flexible, and therefore we were able to build a complete system by only using obvious and isolated landmark descriptors. Fig. 19 shows an example of navigation using the full description language.

5.6.1. Identifying an obvious landmark

Identifying an obvious landmark in a view window is very simple. By definition, an obvious landmark is the only visible object in the window other than the current reference landmark. Therefore, the precondition for identifying an obvious landmark is that exactly two objects are visible. Implementation is trivial.

5.6.2. Identifying an isolated landmark

Our algorithm to compute the most isolated landmark in a scene consisting of n point-like objects uses the concept of *mutual neighborhoods* [9]. (Several other definitions of "isolated" were shown to lead to unstable performance or costly computations.) The algorithm is as follows:

(1) For each pair of objects in the scene compute the *mnv* (mutual neighborhood value). The *mnv* of two objects 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 the second closest object to A, and A is the third closest object to B, then the *mnv* is 5. Note that the *mnv* is symmetric. This result is stored in an $n \times n$ matrix, where n is the number of visible objects. The objects in the left diagram of Fig. 20 have the *mnv* matrix as shown on the right.

(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 corresponds to this particular column) and its "closest" neighbor. Call this value the "c-value" of this particular object. In our example, the c-values for A, B, C, and D are 4, 2, 2, and 3 respectively.

(3) The object (column) that has the largest c-value is the most isolated object in the scene. In our example, A has the largest c-value of 4, and therefore it is the most isolated object. Note that c-value is a small integer of value at most n [20].

6. Error handling

In this section, we focus on various problems that relate to errors that occur before and during navigation. In it, we investigate the sources and magnitudes of navigation error. In addition, and in contrast to Section 2, we no longer assume that the map-maker is mostly error-free and that only the navigator is prone to errors. Due to inherent limitations of sensing elements (in our case, the map-maker's camera and lens), and the dynamics of the environment (lighting, slight displacement of objects in environment, etc.), we relax the error-free assumption of our map-maker. In the beginning of a 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 navigation due to its sensory and motor limitations. When the overall error gets large enough, the navigator is prone to make mistakes. Sometimes this will result in the navigator failing to accomplish its task of reaching the destination. We will analyze each step of this error cascade in turn.

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 helping prevent the navigator from making mistakes. The second method is "error recovery", and is accomplished by the navigator during its navigational task. At each landmark with certain topological properties, the navigator examines its immediate surroundings 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.[7] In this section, we explore both of these in detail.

6.1. Parkway errors

6.1.1. Accuracy of positional information: the map-maker's view

In our physical setting of the map-maker, the global camera is placed at a position that can capture the whole environment, as will be shown in Fig. 24 in Section 7. The global camera passes the positional information of objects to the map-making system through a simple calibration sequence. For simplicity, we assume that the probability distribution of error in each positional information, (x, y), is a Gaussian distribution. We further assume that the standard deviation of the errors in each (x, y) is constant for every object in the environment (these assumptions can be replaced with more accurate ones derived empirically). By a straightforward physical setup, we were able to measure

[7] Our error recovery method can be classified as "forward recovery", as defined in [25], because it replans to get back on track, instead of retracing its traveled route.

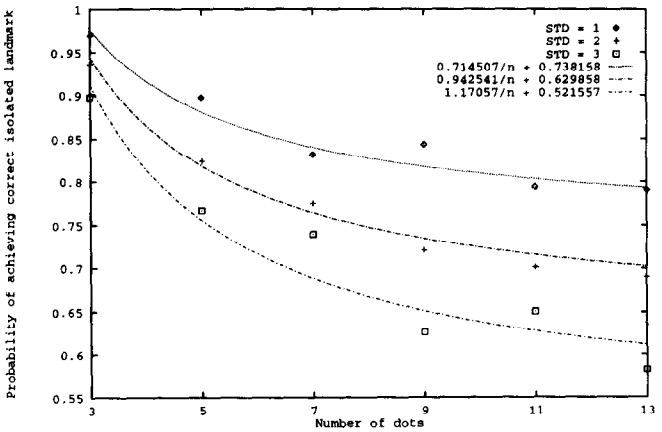


Fig. 21. Reliability of isolated landmarks decreases as the number of neighboring objects increases.

the sample means and the standard deviations of the x and y errors [20], as follows. The units are in millimeters.

$$(\mu_x, \mu_y) = (0.037, 0.122), \qquad (\sigma_x, \sigma_y) = (1.499, 1.066).$$

The significance of this calibration is that specific measurements are now available to the map-maker for use in estimating landmark reliability.

6.1.2. *Reliability of isolated landmarks: the navigator's view*

Results of our experiments using various custom maps, populations, and start and goal positions indicated that the navigator tends to fail in subareas which are highly cluttered. To explain this phenomenon, we resorted to statistical experiments, since the non-linear definition of "isolated" defies easy analytic solutions. After experimentation with several simple functions, we modeled the reliability of the isolated landmark detector with the following function, which states that the reliability is inversely proportional to the number of neighboring objects and directly related to the standard deviation of the position estimation error (σ); this appeared to capture the appropriate behavior well.

$$R(n, \sigma) = \frac{A\sigma + B}{n} - C\sigma + D. \tag{1}$$

This equation was fitted with the data points using the *Mathematica* package [28], and the result is shown in Fig. 21. The significance of Eq. (1) is that the map-maker has an approximate way of predicting the reliability of a landmark, based on the number of landmarks visible in a view window (n), and based on the statistical reliability of the navigator's camera (σ). The reliability can then be translated into some or all of a cost measure for navigation involving each particular landmark in the window.

6.2. *Trajectory errors*

The navigator following a trajectory movement is doing a form of blind spatial search. The navigator has defined the trajectory based on two landmarks in its current

view. Then, it simply moves along this formed direction until a new object appears in its view. The actual distance the navigator must travel to reach the trajectory target, denoted by X_T, may be longer than the estimated distance calculated by the map-maker. Likewise, the distance traveled by the navigator before another (but incorrect) object is reached, denoted by X_e, may be shorter than estimated distance: if $X_e < X_T$, then the navigator has made an error by selecting the wrong landmark. Note that the possibility for error involved in obtaining a wrong object during a trajectory movement is strictly one-dimensional, as opposed to the two-dimensional possibility of error in "the most isolated landmark" method. In the latter case, the neighboring objects in the area surrounding the actual isolated landmark contribute to the error. However, in the trajectory case, the direction of the navigator's trajectory movement is the only source of error: errors perpendicular to the direction have essentially zero effect.

The error in a trajectory task can come from the sensor error in determining the positional information of the landmark objects that define the trajectory direction, or from the translation error or the rotation error of the view window during the trajectory movement. The first of these three sources produce a static error, meaning that the error does not change with the traveling distance of the navigator. But the errors due to the second and the third sources are dynamic (cumulative), meaning that the error typically grows as the trajectory distance increases. Therefore, we modeled the overall error, E, as a Normally distributed function of the map-maker's position estimation error (σ_m), the navigator's position estimation error (σ_n), the navigator's translational error (σ_t), and the navigator's orientational error, as follows [20]:

$$E \sim N\left(0, \sigma_t^2 + (\sigma_m^2 + \sigma_n^2)\left[2\left(\frac{h\cos\theta}{l\sin\theta^2}\right)^2 + 1\right]\right). \tag{2}$$

Here, θ is the incident angle between the trajectory vector and the top leading edge of the sliding window, h is the vertical distance between the anticipated trajectory goal and the leading edge of the view window, and l is the distance between the two objects that define the trajectory (see [20] for details). The significance of this error model is that the map-maker can calculate the reliability of a trajectory.

6.3. Error detection and error recovery

Often it is possible for the navigator to detect, from topological properties of its immediate environment, whether or not it has obtained the landmark intended. Somewhat less often, it is able to recover from a detected error.

6.3.1. Error detection

When a landmark is obtained, the navigator can observe its neighborhood in order to verify that it is on the correct course. Usually, the current view of the navigator alone does not contain enough information for verification. Only sometimes it does: if the navigator has incorrectly sought and adjusted to a landmark, but realizes that no other landmarks are visible in the window, this is clearly an error, since further progress is impossible.

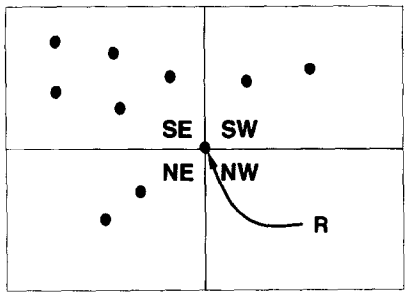


Fig. 22. Example configuration for the Observe( ) function, showing four different window positions pivoted on the reference landmark.

To verify in the more usual cases, a wider environmental context is necessary. The largest region that the navigator can observe without losing its current reference landmark is the area covered by "looking around" its current reference landmark in the following way. Suppose the reference landmark is at the SE corner of the window; call the corresponding area covered by the window the SE quadrant. By moving itself so that the reference landmark appears in the other three distinguished locations of its window (SW, NE, NW), a context four times as large becomes visible, but without loss of recoverability of the current location. Mistakes or errors can then be detected by observing the four quadrants relative to the current reference landmark and analyzing whether the visual properties in these area of space are correct.

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

signature =
{Observe(SE), Observe(SW), Observe(NE), Observe(NW)},

where Observe(Q) is a compact topological description of the corresponding quadrant.

If the signature of an improperly attained landmark is different from the signature of the intended landmark, 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. Implementation of the Observe( ) function depends on the capabilities of the navigator. For our simple navigator model, we developed several Observe( ) functions, of which the simplest is the QOC (Quadrant Occupancy Count). This is defined to be a sum of four binary integers indicating the presence or the absence of objects in each quadrant. Thus, $0 \leqslant \mathrm{QOC} \leqslant 4$. In Fig. 22, the QOC of object R is 3, because 3 of the quadrants "looking around" R are occupied.

Errors can be detected only if the landmark has a unique QOC within the current view window. In Fig. 23, we see there are 5 landmarks other than the current reference landmark in the view window. R is the current reference landmark and I is the desired isolated landmark to be sought. The QOC of the visible objects are shown in Table 3. As can be seen, the QOC of object I is unique within this view window, thus an error can be detected by examining the QOC of the obtained landmark; any value other than

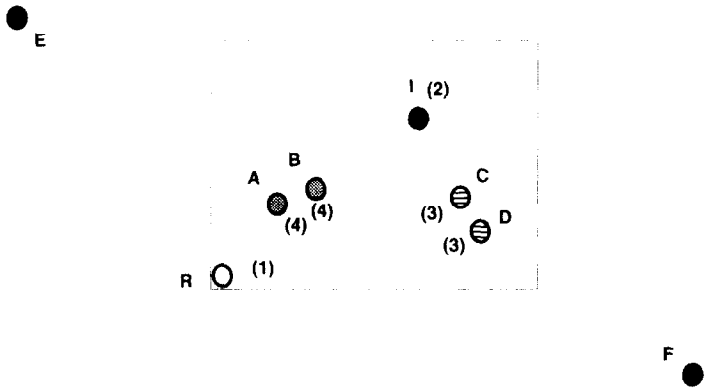


Fig. 23. Error-detectable and error-recoverable configuration.

Table 3
QOC values of objects A, B, C, D, I and R

A	B	C	D	I	R
4	4	3	3	2	1

2 indicates an error. Note that object E affects the QOC of objects A and B, and object F affects the QOC of object D.

6.3.2. Error recovery

Error detection is useful only if there are corresponding error recovery methods available to the navigator. In the worst case, when an error is detected the navigator can simply give up its navigational task and effectively say "I am lost". (What happens next is beyond the one-way communication model assumed in this work.) But in many cases, by performing a fixed set of "reset" sequences, the navigator is able to get back to its originally intended course. In this subsection, we describe such an error recovery routine that the navigator can execute when errorful situations are encountered.

The basic idea is to assign a fixed action for each detected QOC (0 to 4). Therefore, the error recovery is a simple "if QOC then ACTION" set. For example, in Fig. 23, objects A and B have a QOC of 4. If the navigator finds itself at either A or B instead of the desired I, it can "correct" the mistake by moving A or B to the SW corner and then again selecting the isolated landmark. Likewise, the correction instruction for objects C or D whose QOC is 3, is (SE, Isol), namely, move C or D to the SE corner and seek the most isolated landmark.

What is surprising about this figure is that landmark I is guaranteed to be properly attained, since I has a unique signature, and all other landmarks that can be mistakenly attained have recovery actions that correct the navigator towards I. Even if these corrections mistakenly result in some landmark other than I being attained, repeated applications of the "If QOC then ACTION" direction eventually result in I being attained, as it is uniquely recognized by its QOC.

Formally, we call an isolated landmark context to be error-recoverable if:

(1) The configuration is error-detectable (the isolated landmark has a unique QOC).
(2) Objects with the same QOC are positioned so that the originally intended landmark (the true isolated landmark, I) is in the same quadrant relative to these objects. For example, in our example of Fig. 23, objects A and B have QOC 4, and both have I visible from the SW quadrant.
(3) From every reachable object which can be mistaken as the true isolated landmark, there must exist a path to the true isolated landmark. For example, in our Fig. 23, the isolated landmark for object C from the SE quadrant is in fact object B, not I. But I is reachable from B. Therefore, I is transitively reachable from C.

6.3.3. *Error detection and error recovery in trajectories*

Similarly, we can use the QOC methods to detect and to recover from errors in trajectory tasks by appropriately modifying the existing algorithms for parkways.

6.3.4. *Example*

Let us refer back to Fig. 14. The corresponding custom map for this path is:

```
((Isol,SE), (Isol,NE), (Isol,NE),
 (Isol,Trajp,(2,NW,Isol),(1,-),-))
```

There are four entries in the custom map. First three take the navigator from the starting position (shown on the lower right-hand corner in the figure) to the end of the first parkway. There is no error recovery sequences embedded in the first three entries, but for valid reasons. In the first and the third entry commands, neither error recovery nor error detection is possible; the landmarks in the corresponding windows do not have unique QOCs. In the second entry, the navigator sees only one object (the obvious landmark) in its view; therefore, error verification is implicit and error recovery is not necessary. In the last entry of the custom map, the navigator is told to identify the isolated landmark and to move in the positive direction of the formed trajectory. If the newly appearing object has QOC of 2, then the navigator is told to move to the NW and to pursue the isolated landmark. Otherwise, if the QOC is 1, it has reached the destination.

7. Implementation and experiments

7.1. *Implementation*

We have implemented our map-maker and navigator using an IBM 7575 SCARA robot arm, two Sony XC-77 CCD cameras, a PIPE (which is a high speed real-time image processor), and a Sun SPARC workstation for high-level control of the navigator and the map-maker. Fig. 24 shows the configuration of the map-maker and the navigator.

7.1.1. *The map-maker and the navigator*

The map-maker is comprised of a CCD camera located at a position that can capture the whole workspace of the navigator. This camera is attached to the PIPE, which grabs

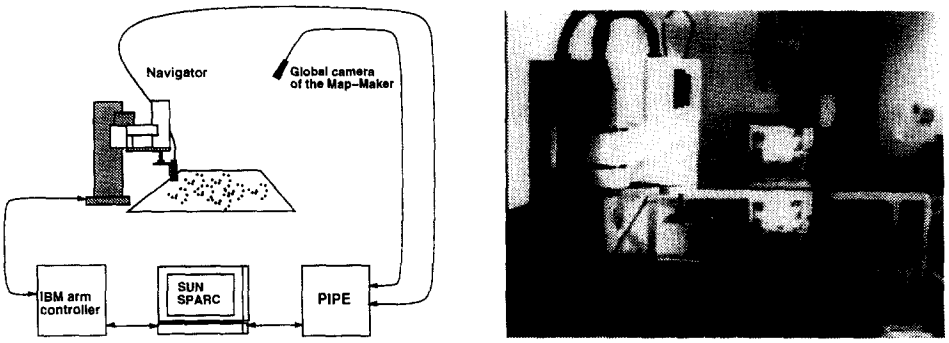


Fig. 24. The experimental setup configuration.

the image and sends it to the Sun workstation, which runs the "map-making" program based on the centroid information of the scattered objects. The assumption of near omniscience in the map-maker requires that the image captured by the global camera (see Fig. 24) be used correctly 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 $(U, V, 1)$ [7]. The map-maker generates a file called "custommap", which contains the list of directions for the navigator.

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 direction in the custom map, the detection of the most isolated landmark and the amount of "adjusting" of the robot for the corresponding direction is computed by the Sun workstation. It then sends out low-level instructions to the IBM arm controller for the actual movement.

7.1.2. The user interface program

The user interface program of the map-maker for the purpose of interaction with a human experimenter is implemented on the Sun workstation, running an X-Window system. When the map-maker program is started, the computed position of each scattered object in the environment is collected and displayed on the interface windows. Using available command buttons, such as "start", "goal", "map", "clear", and "quit", the user is able to choose the initial position and the destination, and to ask the map-maker to compute the desired type of optimal path and to generate custom map. Fig. 25 shows the user interface program.

7.2. Experiments

7.2.1. Sparse versus densely populated environments

As described in Section 6, our statistical experiments indicated that the isolated landmark descriptor is less reliable in a scene with a dense population. We also noted that the reliability is dependent on σ, which measures the accuracy of the sensor. Our

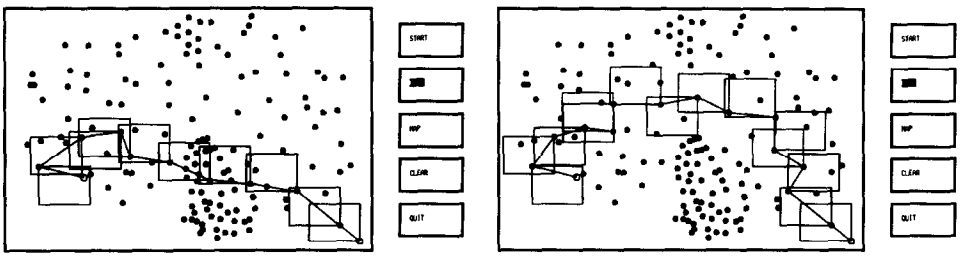


Fig. 25. An example showing a path that passes through a highly cluttered unreliable region (left), and the result of using the "reliability" criteria to avoid the cluttered regions (right).

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 Fig. 25 shows a path created by the map maker solely using distance as a definition of cost—the path passes through the "wall" of a highly cluttered region of the environment, where it sometimes becomes confused. On the right, we see an adjusted path that avoids the unreliable cluttered region by incorporating reliabilities in the computation of the custom map.

7.2.2. *Cost optimality criteria*

Navigation can be done with various costs; we have experimented with several.

- *Minimum travel distance*: Minimizing the travel distance is done in terms of the context-based distance as described in Section 5. The reliability of the shortest distance path is sometimes very low when there are cluttered regions along the way. We saw such a path in the right diagram of Fig. 14 in Section 5. The corresponding costs for this path are:

```
Distance cost       : 444.930
Unreliability cost  :   0.539
```

  The unreliability cost is measured by $-\log(R)$, where R is the reliability. We use the unreliability measure instead of R for the purpose of deriving the path using Dijkstra's shortest path algorithm, which minimizes cost.
- *Highest reliability*: Maximizing the reliability (or minimizing the unreliability) of the overall path usually yields a longer distance path as described in Section 5. We see such a reliable but detouring path in the right diagram of Fig. 14 in Section 5. The corresponding costs for this path are:

```
Distance cost       : 731.506
Unreliability cost  :   0.127
```

- *Hybrid method*: The use of a compromised cost function to minimize the D/R ratio, as described in Section 5, yields a path that is a neither too long nor too unreliable. We see a such path in Fig. 15. The corresponding costs for this path are:

```
Distance cost       : 457.931
Unreliability cost  :   0.214
```

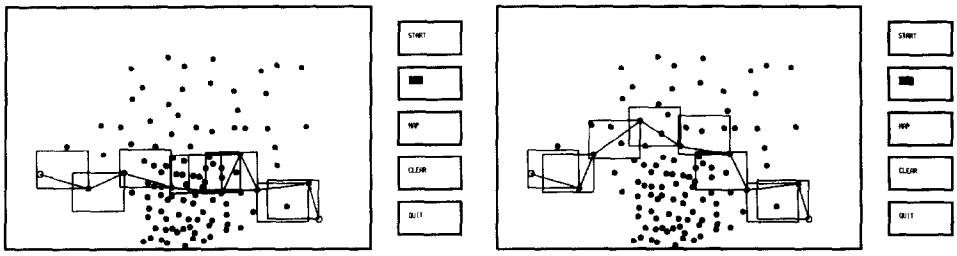


Fig. 26. The D/R path when σ is small (1 mm) is shown on the left, whereas the D/R path when σ is large (5 mm) is shown on the right.

As we can see, the shortest distance path has the lowest reliability (highest unreliability measure), whereas the most reliable path has the highest cost in terms of the distance travel. The compromised path has intermediate cost values in both measures. Other costs can be taken into account; in particular, the length of the direction sequence can be considered a cost to minimize. Such an incorporation is particularly straightforward; each transition only has unit cost.

7.2.3. Effects of sensor accuracies

Accuracy of the map-maker's and the navigator's sensors are modeled with the positional information error estimates as presented in Section 6. Depending on the standard deviation of the positional error, which is directly related to the sensor accuracy, the reliability of each path segment may differ. In this subsection we compare paths generated at different sensor accuracies (σ).

- *Small σ (accurate sensor)*: As σ gets small, the reliability of an isolated landmark increases. The left diagram of Fig. 26 shows a path generated by the map-maker using the compromised cost function D/R, when σ is equal to 1 mm. Because R is large (high reliability), the computed path is close to the shortest distance path. Note that the navigator passes through the cluttered region. The corresponding costs for this path are:

```
Distance cost       : 420.57
Unreliability cost  :   0.91
```

- *Large σ (inaccurate sensor)*: As σ gets large, the reliability of an isolated landmark decreases, as shown in Fig. 21. The right diagram of Fig. 26 shows a D/R path generated by the map-maker when σ is equal to 5 mm. Because R is small (low reliability), the computed path resembles more the most reliable path. Note that the navigator tries to avoid the cluttered region. The corresponding costs for this path are:

```
Distance cost       : 460.43
Unreliability cost  :   2.27
```

7.2.4. Context-based navigation

As previously stated in Section 5, the distance between landmarks is not necessarily equal to the actual travel distance. In an extreme case, as in Fig. 16, we saw that the

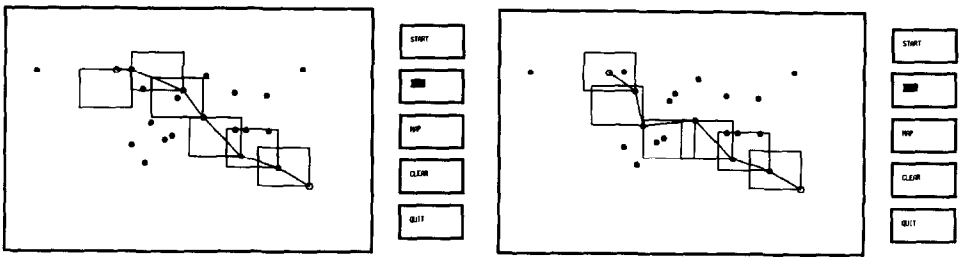


Fig. 27. The shortest landmark distance path (left) and the shortest context-based path (right).

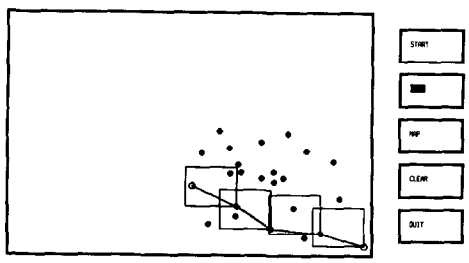


Fig. 28. A simple error-recoverable path (path starts in the lower right).

context-based shortest path can be a zigzag type of landmark path. The diagram in the left of Fig. 27 shows a shortest path generated in terms of minimizing the distances between the landmarks. On the right, we see the shortest context-based path for the same start-goal configuration. The landmark-based distance cost and the context-based travel distance cost for the path in Fig. 27 are:

```
Landmark-based distance cost: 385.465
Context-based distance cost:  358.639
```

The landmark-based distance cost and the context-based travel distance cost for the path in the right diagram of Fig. 27 are:

```
Landmark-based distance cost: 411.413
Context-based distance cost:  347.039
```

Intuitively, in the right diagram, there is more "sliding" of the navigator, particularly between the third and the fourth view window.

7.2.5. Error detection and recovery

We have implemented the QOC method as described in Section 6. In Fig. 28, we see a path generated in a simple environment. The starting position is at the lower right-hand corner. The corresponding custom map for this path is:

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

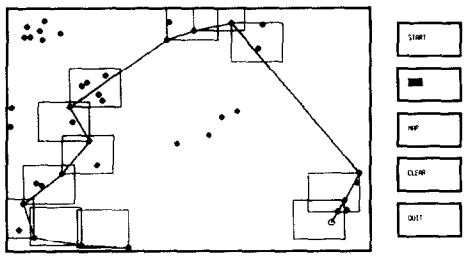


Fig. 29. A highly error-recoverable path (path goes clockwise).

In our test run with the robot, the navigator failed to achieve 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. Therefore, the navigator incorrectly identified the top-most object in the window as the isolated landmark. When the error verification sequence was executed, the QOC was 4, which indicated to the robot to adjust to NE and to recapture the correct landmark. In this case, the error recovery routine correctly handled the error. After the error recovery routine, the navigator attained the destination directly and successfully. Note that this journey would have been a failure if not for the error recovery routine. Effectively, the error recovery routine has increased the reliability of this path.

Fig. 29 shows a path generated by the map maker that is highly error-recoverable. The navigator starts near the lower left-hand corner of the environment and navigates to the destination at the lower right-hand corner. This path consists of both parkway path segments and trajectory path segments that are error-recoverable. Corresponding commands in the custom map (as generated on the user interface displayed during the map-making session), including the error detection and error correction statements, are shown in Fig. 30.

7.2.6. *The definition of success and failure*

The navigation can either be a success, in which case the navigator correctly reaches the destination, or be a failure, in which case the navigator ends up in a wrong place. However, not all successes and failures are equal. We have categorized them as the following.

Types of success

(1) The navigator follows the correct path and reaches the goal directly. The navigator may or may not be equipped with error handling routines. If the navigator is provided with error handling routines, all landmarks are verified correctly during the navigation without having the need to execute recovery sequences. The navigator is "aware" of the success, that is, it can report that it has attained the goal.

(2) The navigator follows the correct path and reaches the goal using error recovery routines. The navigator is aware of the error corrections, and of the success of the journey.

```
FROM  TO CMD    DISTANCE  LOGPROB  COST
 90   42 ISOL    78.102   -0.000   78.102
         VERIFY QOC -> 2 (TRIVIAL CASE)

 42   30 ISOL    80.231   -0.000   80.231
         VERIFY QOC -> 2 (TRIVIAL CASE)

 30   27 ISOL    88.865    0.024   88.865
         VERIFY QOC -> 3 (CORRECTABLE)
            If QOC -> 2 (SW, ISOL)

 27   55 ISOL    81.216    0.087   81.216
         VERIFY QOC -> 2 (CORRECTABLE)
            If QOC -> 3 (SW, ISOL)

 55   58 ISOL    68.600    0.024   68.600
         VERIFY QOC -> 3 (CORRECTABLE)
            If QOC -> 2 (SE, ISOL)

 58    3 ISOL    77.104    0.024   77.104
         VERIFY QOC -> 2 (CORRECTABLE)
            If QOC -> 3, (SE, ISOL)

  3   87 TRAJP  226.500    0.350  226.500
         VERIFY QOC -> 1 (CORRECTABLE)
            If QOC -> 2, (NE, ISOL)

 87   79 ISOL    47.434    0.024   47.434
         VERIFY QOC -> 3 (CORRECTABLE)
            If QOC -> 2, (NW, ISOL)

 79   81 ISOL    81.688   -0.000   81.688
         VERIFY QOC -> 2 (TRIVIAL CASE)

 81  100 TRAJP  402.040    0.268  402.040
         VERIFY QOC -> 1 (CORRECTABLE)
            If QOC -> 2, (SW, ISOL)

100  164 ISOL    51.478    0.127   51.478
         UNVERIFIABLE (Not Correctable)

164  108 ISOL     0.000    0.024    0.000
         VERIFY QOC -> 1 (CORRECTABLE)
            If QOC -> 2, (NE, ISOL)
Overall distance is 1283.258911
Overall -Log reliability is 0.951362
run>
```

Fig. 30. Custom map commands corresponding to the path shown in Fig. 29, as shown to the user during the map-making session.

(3) The navigator follows the wrong path, but ends up correctly. In this case, the navigator has made incorrect moves, incorrect validations, or incorrect recoveries, but somehow ends up in the right place. The navigator is aware of the success, but is unaware of its mistakes.

Types of failure

(1) The navigator stops due to inability to proceed because it is seeking a landmark, but is unable to locate it. For example, the custom map instructs the navigator to identify an obvious landmark, but there are more than one landmarks visible (so there is no obvious landmark). The navigator is "lost" and the navigation has failed. The navigator is aware of the failure. Note that error detection or error correction cannot help, as the error is in the description, not the action.

(2) The navigator stops due to inability to proceed because it is seeking a landmark that is error-detectable, but not error-recoverable. Had the correct landmark been verified, the navigator would have been able to proceed. But if the error verification fails, the navigator is unable to go on because an error recovery sequence is unavailable. The navigation is aware of the failure. Note that the navigator is lost, but in a different sense of the word: it has verified that it is lost, but doesn't know how to correct itself.

(3) The navigator follows an incorrect path to its end which is not the goal landmark. In this case, the navigator is unaware of its mistakes. The navigation has failed, but the navigator is unaware of the mistakes and is also unaware of the failure; it reports success.

8. Summary and conclusions

We have discussed a new method for topological navigation in a large unstructured environment, where metric information is unreliable and expensive [11, 13, 26]. Most existing qualitative robot navigators emphasize the importance of landmarks. Their navigation depends heavily on the usage of visual landmarks, however, with assumptions that landmarks can be readily identified. We argue that without a detailed criteria for selecting good landmarks, topology-based navigation is not possible.

We started out by asking several important questions concerning landmarks, such as (1) what is a good landmark, (2) what features of a landmark are important, (3) what sensors are to be used to recognize a landmark efficiently, (4) how to describe a landmark to the navigator, (5) how to detect errors, and (6) how to recover from them. During our efforts to answer these questions, further interesting and important issues were brought up, such as (1) the roles of the direction giver and the direction follower for navigation, (2) the lower bounds of the navigator's intelligence and its ability for efficient and accurate navigation, (3) the quantification of environments that are good or bad for navigation, (4) qualitative methods that can characterize landmarks so that a navigator with low metric ability can correctly identify them, (5) the formal definition of a landmark and the assigning of a "goodness" measure to it, and (6) error prevention, detection, and recovery techniques.

For the purpose of efficiency and of conceptual modularity, we suggested the dichotomy of the navigation system. The first module is the map-maker (the direction giver), who has an accurate knowledge of the environment and of the navigator. It abstracts the world into data structures that are mostly topological. The map-maker computes paths based on the configurations of visible landmarks, and generates customized directional instructions for the navigator in a qualitative and efficient way, all prior to the actual navigation. The second module is the navigator (the direction follower), who is assumed to have a very limited sensor range and almost no metric capabilities. Its objective is to get to the destination by executing each of the provided directional instructions in a sequence. The navigator's view of the world is, therefore, an even more abstracted one. We used a very simple navigator model as the direction follower for the sake of the efficiency during traversal (the less cognitive strain, the faster). Most of the difficult and time-consuming reasoning is done by the map-maker in the pre-processing stage. During the navigation, because a robot must act in real time relative to a task environment, it has limited time to assimilate its observations to perform appropriate actions. At the same time, its computing capacity and the working memory must be shared with other pressing tasks. A graphical representation of the relationships between the world, the map-maker, and the navigator is shown in Fig. 31.

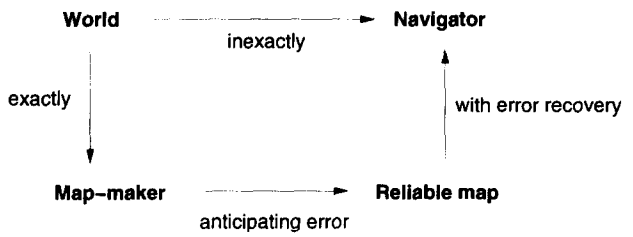


Fig. 31. Abstraction of the world for the map-maker and the navigator.

We introduced abstract data structures called parkways and trajectories which are abstractions of the real world into topological path representations. Parkways and trajectories enabled us to define cost measures of the travel route, and to quantitatively evaluate its goodness. We presented a lower-level abstraction of the world, as the navigator experiences it. We suggested the development of a description language to identify landmarks qualitatively for robustness and efficiency. We provided a basic description language for our experimental world, and gave details in the implementation of the isolated landmark descriptor. We analyzed and modeled various sources of errors, thereby assigning the "goodness" of each landmark based on the computed reliability and reachability. We studied error detectability and recoverability in both parkway and trajectory paths, and proposed general recovery schemes. As a special case, we analyzed in detail and implemented error detection and error recovery methods that make use of the Quadrant Occupancy Count (QOC). We discussed path planning and custom map generation. We observed the relationship between paths (parkway and trajectory paths) and the corresponding transition matrix entries. The transition matrix provided us with a computational definition of a landmark: its columns were mostly filled; the associated cost in each transition matrix entry defined the "goodness".

We have implemented a demonstration system using a camera mounted robot arm that navigates over a two-dimensional planar world with point-like objects randomly placed on it. The use of point-like objects forces us to define a landmark with respect to its neighboring objects, rather than by its intrinsic qualities. The freedom to place the objects randomly eliminates the feasibility of using external structural cues (e.g. compass, roads) for navigation. We presented the implementation and the experimental results of this demonstration system.

To emphasize the topological aspects of navigation, we designed a highly abstract environment. For more realistic applications that bring the level-helicopter scenario more down to earth, we note that under many other models of navigation it is still the case that many of these concepts remain, although sensing costs increase and some additional phenomena need to be addressed.

Perhaps the most abstract model of on-land vehicle navigation related to our work would model the world surface as a plane, would have objects of fixed size and shape (this makes the task harder, in the sense that descriptions remain purely topological), and would have a fully panoramic retina (that is, the navigator can pan 360 degrees). In this model, occlusion of objects by other objects is possible, and the perspective effects of the camera require some cutoff for the sensing of distant objects (otherwise, there is

often no problem, since the goal is visible and needs only to be described). Under this scenario, the definitions of obvious and isolated points remain the same, the definition of "seeking" becomes simply the heading of the navigator in the direction of the target, and "adjusting" becomes defined as running over (or just barely missing) the target. Parkways are defined under this new definition of connectedness, but are qualitatively the same (although most parkways become directed): an object is reachable from a landmark if it is not occluded and is within range in the panoramic view. Trajectories are defined exactly analogously to the level-helicopter case. Error detection becomes more elaborate: there is no QOC, as there are no quadrants, but the vocabulary of the map-maker can be used to describe, in a fixed order from the heading direction, the topology of the sensed environment about the landmark just obtained: instead of "3", it could be a list, like "isolated point, isolated pair, isolated point, large group, isolated point, etc." Error recovery, however, is nearly identical, but may not be as effective: the navigator selects the described object (if it is indeed possible to describe it) and directly adjusts to it.

Other models of the world would progressively relax some of these assumptions further. For example, the camera model of the land navigator may have a limited field of view; this wedge-shaped view of the world would require different abstractions, particularly if the camera were to pan independently of the direction of the vehicle itself. A non-planar world, in which the world surface itself can cause occlusions, would also further complicate the representations of what we called the "context" of a landmark: what is visible would then depend on an accurate modeling not only of the camera capabilities, but of the local topography as well. The last of these assumptions is the most challenging, since topographic properties of the world are not sensible by the map-maker, as we have modeled it. Nevertheless, it is likely that the core concepts of this work, namely, the dichotomy between the map-maker and the navigator, the vocabulary and the grammar of a custom map, the ideas of parkways and trajectories, and the necessity and means of detecting and correcting errors, all based on topology rather than metric values, will remain.

As a whole, we feel that we have provided enough evidence for our belief that large-scale navigation should be more focused on high-level qualitative reasoning. In our experiments, we showed that high-level topological information of the environment is efficient and accurate in our simplified setting. This does not imply that qualitative information is the only useful tool in navigation in general. Whenever metrically accurate information is available, the navigator should use it, but only if the cost can be justified. There is much to be done for this framework to be blended into a real environment. But we claim that the multi-level abstraction of the world offers an effective guideline for the acquisition and organization of knowledge, and its manipulation for more robust and efficient navigation in the large.

References

[1] A. Aho, J. Hopcroft and J. Ullman, *The Design and Analysis of Computer Algorithms* (Addison Wesley, Reading, MA, 1974).

[2] R.A. Brooks, Solving the find-path problem by good representation of free space, in: *Proceedings AAAI-82*,Pittsburgh, PA (1982) 381–387.
[3] R.A. Brooks, A robust layered control system for a mobile robot, *IEEE J. Robotics Automation* **2** (1) (1986) 14–23.
[4] W.G. Chase, Spatial representation of taxi drivers, in: D.R. Rogers and J.A. Sloboda, eds., *The Acquisition of Symbolic Skills* (Plenum, New York, 1982).
[5] J.L. Crowley, Navigation for an intelligent mobile robot, *IEEE J. Robotics Automation* **1** (1) (1985) 31–41.
[6] D. Dai and D.T. Lawton, Range-free qualitative navigation, in: *Proceedings IEEE Conference on Robotics and Automation* (1993).
[7] K. Fu, R. Gonzalez and C.S.G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence* (McGraw Hill, New York, 1987).
[8] G. Giralt, R. Sobek and R. Chatila, A multi-level planning and navigation system for a mobile robot, in: *Proceedings IJCAI-79*, Tokyo (1979) 335–337.
[9] K.C. Gowda and G. Krishna, Agglomerative clustering using the concept of mutual nearest neighborhood, *Pattern Recognition* **10** (1978) 105–112.
[10] J.R. Kender and A. Leff, Why direction-giving is hard: the complexity of linear navigation by landmarks, *IEEE Trans. Syst. Man Cybern.* **19** (6) (1989) 1656–1658.
[11] J.R. Kender, I.P. Park and D. Yang, A formalization and implementation of topological visual navigation in two dimensions, in: *SPIE International Symposia* (1990).
[12] B.J. Kuipers, Modeling spatial knowledge, *Cognitive Sci.* **2** (1978) 129–153.
[13] B.J. Kuipers and Y.T. 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).
[14] T. Levitt and D.T. Lawton, Qualitative navigation for mobile robots, *Artif. Intell.* **44** (3) (1990) 305–360.
[15] T. Lozano-Perez, Automatic planning of manipulator transfer movements, *IEEE Trans. Syst. Man Cybern.* **11** (10) (1981) 681–698.
[16] V. Lumelsky and T. Skewis, A paradigm for incorporating vision in the robot navigation function, in: *Proceedings IEEE International Conference on Robotics and Automation* (1988) 734–739.
[17] D.M. Mark, Finding simple routes: 'ease of description' as an objective function in automated route selection, in: *Proceedings Second IEEE Conference on Artificial Intelligence Applications* (1985) 577–581.
[18] D.M. Mark, On giving and receiving directions: cartographic and cognitive issues, in: *Proceedings Auto-Carto 8* (1987).
[19] H.P. Moravec and A. Elfes, High resolution maps from wide angle sonar, in: *Proceedings IEEE Conference on Robotics and Automation* (1985) 116–121.
[20] I.P. Park, Qualitative environmental navigation: theory and practice, Ph.D. Thesis, Columbia University, New York (1993).
[21] C.K. Riesbeck, 'You can't miss it': judging the clarity of directions, *Cognitive Sci.* **4** (1980) 285–303.
[22] M.H. Soldo, Reactive and preplanned control in a mobile robot, in: *Proceedings:Image Understanding Workshop* (1990).
[23] L.A. Streeter and D. Vitello, A profile of drivers map-reading abilities, *Human Factors* **28** (2) (1986) 223–239.
[24] L.A. Streeter, D. Vitello and S.A. Wonsiewica, How to tell people where to go: Comparing navigational aids, *Int. J. Man-Mach. Stud.* **22** (1985) 549–562.
[25] E. Stuck, Detecting and diagnosing mistakes in inexact vision-based navigation, Ph.D. Thesis, University of Minnesota, Minneapolis, MN (1992).
[26] K.T. Sutherland and W.B. Thompson, Inexact navigation, in: *Proceedings IEEE Conference on Robotics and Automation* (1993).
[27] P. Thorndyke and B. Hayes-Roth, Differences in spatial knowledge acquired from maps and navigation, *Cognitive Psychol.* **14** (1982) 560–589.
[28] S. Wolfram, *Mathematica* (Addison Wesley, Reading, MA, 1988).
[29] J.Y. Zheng and S. Tsuji, Panoramic representation for route recognition by a mobile robot, *Int. J. Comput. Vision* **9** (1) (1992) 55–76.