[8] Jr. H. A. Gleason. *Linguistics and English Grammar*. Holt, Winehart, and Winston, NY, 1979.

[9] Monnett Hanvey. Enviromental representation for mobile robot navigation. Technical report, Department of Computer Science, Columbia University, January 1988.

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

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

[12] Richard C. Karp. *Reducability Among Combinatorial Problems*, pages 85–103. Plenum Press, New York, NY, 1972.

[13] John R. Kender and Avraham Leff. Why direction-giving is hard: The complexity of linear navigation by landmarks in one-dimensional navigation. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6):1656–1658, November/December 1989.

[14] Benjamin Kuipers. Modeling spatial knowledge. *Cognitive Science*, 12:129–153, 1978.

[15] Massachusetts Computer Corp. *MC5600/5700 Installation Guide, Part Number 075-04007-00-0 (Revision A)*, April 1986.

[16] Massachusetts Computer Corp. *UNIX Programmer's Manual, Part Number 075-01012-00-0 (Revision K)*, February 1988.

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

[18] L. A. Streeter, D. Vitello, and S. W. Wonsiewicz. How to tell people where to go: Comparing navigational aids. *Int. J. Man-Machine Studies*, 22(5):549–562, May 1985.

apply this n times. The second method uses n cost matrices, but computes the optimal paths in a single pass. The search is done using the primary cost matrix and when there is a tie, the secondary cost matrices are considered. This second method increases matrix computation time and space to save search time. A third method uses a single compressed cost matrix and a single-pass. Each entry in the cost matrix is an algebraic combination of primary and secondary costs, encoded in odometric fashion. A single pass of search would suffice. However, one drawback of this method is that if the number of optimality criteria is very large, we would have to use a very large number of bytes (or words) to hold each cost matrix entry; further, the computation involving thes numbers would also be costly.

6 Conclusion and future work

Navigation-in-the-large addresses some of the fundamental issues of AI in computer vision. Representations of objects with an premium on uniqueness, heuristics for finding good visual landmarks quickly, and the design of fall-back wandering scans of an environment all appear necessary. With the need to handle object occlusions or navigator oblique views, rules of thumb become paramount. We hope the end result of these investigations would be a greater understanding of how to quantify the visual difficulty of a world, and how to quickly determine what visual features or viewing strategies would provide effective custom maps for navigation.

References

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

[2] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

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

[4] Ernest Davis. *Representing and Acquiring Geographic Knowledge.* Morgan-Kaufmann, 1986.

[5] Thomas Dean. A taxonomy of map learning problems. Technical Report CS-87-16, Department of Computer Science, Brown University, August 1987.

[6] R. J. Elliott and M. E. Lesk. Route finding in street maps by computers and people. In *Proceedings of the 1982 AAAI Conference*, pages 258–261. IEEE, 1982.

[7] Michael R. Garey and David S. Johnson. *Computers and Intractability.* W. H. Freeman, 1979.

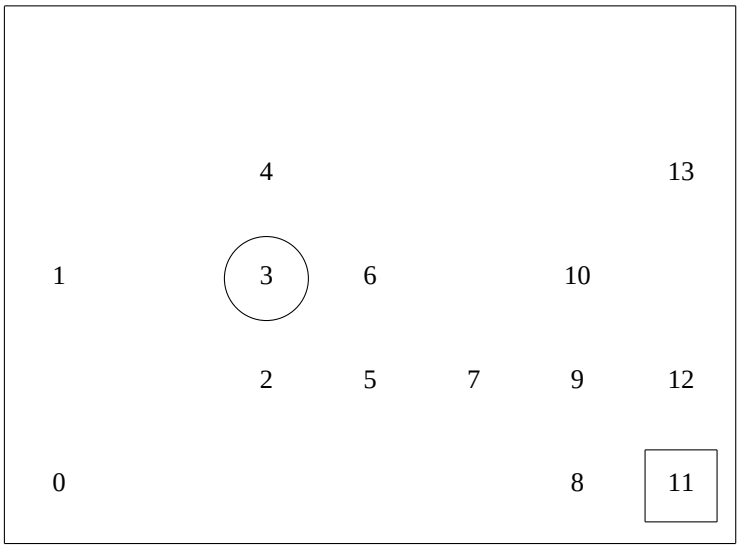

Figure 3: Labeling of landmarks in the sample dots world

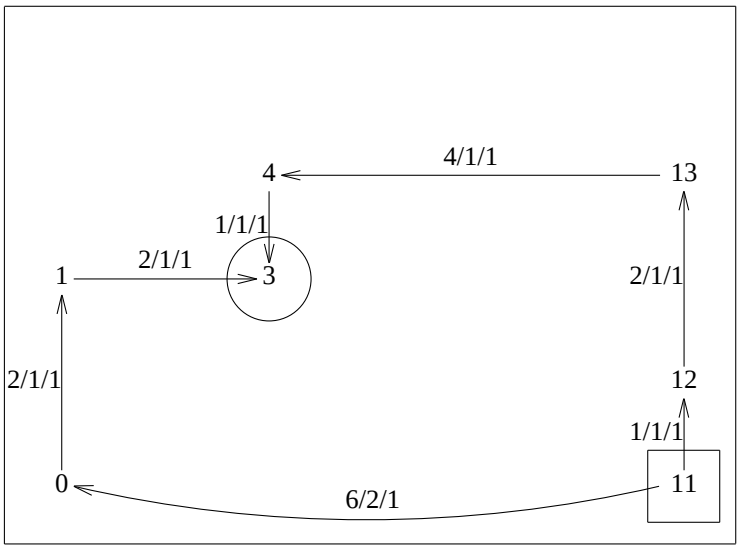

Figure 4: Paths generated by map-maker

triangular, which eventually fills a specified quarter of the plane, or semi-triangular, which fills an eighth.

Although it is difficult to analytically determine the expected number of highways given a specified object density, a series of experiments suggested that the number of highways is at a maximum when object density is about .03. Below .03, objects tend to be completely isolated; above .03, highways start to link up into one world-wide highway. An analytic proof would have some of the flavor of the proof of the birthday paradox: to start a new highway, each additional object would have to avoid all existing highways, which becomes increasingly difficult.

5 Experimental results

For our experiments, we have allowed one primary cost optimality criteria and one secondary criteria. Six different paths were possible: sensor-distance, sensor-maplength, distance-sensor, distance-maplength, maplength-sensor, and maplength-distance. The difficult part was coming up with a set of landmark placements so that the paths generated by the map-maker are as distinct as possible. There seemed to be no apparent heuristic to do so, although it is clear that $m \leq s \leq d$. Using the landmark layout in Figure 3, we have the following paths:

```
d-s : 11  8  9  10  6  3   (d=6, s=5, m=5)
d-m : 11 12  2   3         (d=6, s=6, m=3)

s-d : 11 12 13   4  3      (s=4, d=8, m=4)
s-m : 11  0  1   3         (s=4, d=10, m=3)

m-d : 11 12  2   3         (m=3, d=6, s=6)
m-s : 11  0  1   3         (m=3, d=10, s=4)
```

Figure 4 shows two of the optimal paths (s-m and s-d) generated by the map-maker. The numbers on the arcs indicate costs $d/s/m$.

The algorithm to derive optimal paths that satisfies the primary and the secondary criteria had to be carefully designed in order to prune the exponentially large number of paths created. There are several ways to solve this problem. The first is a n-stage search method, where n is the number of optimality criteria being used. Using the primary criteria, the algorithm creates a tree of primary optimal paths, then in the next round of search, considers only the edges that appeared on the primary paths. If we have n optimality criteria, we

When every pair of landmarks has been examined for reachability, we can produce sets of connected components. We define such a set as *highway*. For a given world, multiple highways can exist. Thus, the navigator can travel from one landmark to another only if these two landmarks are on the same highway. Computing *highways* can be done in $O(M + N + K)$, where M and N are the row, column sizes of the grid world and K is the number of landmarks populated. In $O(M + N)$ we bucket sort the cells, once as (i, j) and once as (j, i). Then we have only to find the edges between adjacent points in rows and columns. Since we have K landmarks, the result is a graph with K vertices and $O(K)$ edges; in $O(K)$ time we find its connected components. This implies that it takes $O(M + N + K)$ to find out whether the goal is a place that "you can't get to".

There are two main data structures used by the map-maker module, the transition matrix and the cost matrix. The entry (i, j) in the transition matrix is filled with a direction - *north, east, south* and *west* - and a descriptive feature value of the landmark l_j if it is reachable from landmark l_i. For example, if l_j is at relative north from l_i and its south edge has value 7, i.e., three dots in a row, then the transition matrix entry (i, j) would be (*north*, 7). The cost matrix contains the cost of such a transition. In our model, we computed three different optimality criteria: sensing cost (s), distance traveled (d), and the map length (m). Distance traveled is defined as the number of cells the navigator travels over. Sensing cost is the number of landmarks encountered on the way. Map length is the number of instructions given to the navigator.

The map-maker computes the optimal route, based on the optimality criteria, by Dijkstra's shortest path algorithm. The time complexity for Dijkstra's algorithm is $O(n^2)$ if no fancy data structure is used [2], but if a Fibonacci heap is used as a priority queue, Dijkstra's algorithm takes $O(e + \log n)$, where e is the number of edges and n is the number of vertices.

Another way of finding the optimal path is to use a heuristic search algorithm such as branch-and-bound or the A^* algorithm [17]. These algorithms are more costly than Dijkstra's algorithm in the worst case, but if the heuristic function is "good", they can be very effective. We found that in the case of sensing cost (s) estimation or map length (m) estimation, the underestimating heuristic that assigns a cost of 1 to objects in the same row/column, and 2 elsewhere occasionally gives dramatic improvement. Also in the case of distance cost (d) estimation, using the Manhattan distance metric between current position and the goal as underestimating heuristic gave us good results. We are attempting to quantify under what circumstances heuristic A^* outperforms Dijkstra.

Note that optimal paths can be computed only if the start and goal landmarks are on the same highway. Inter-highway jumping methods are possible, but costly and possibly inaccurate. We have defined several *wandering* strategies, which are basically a space-filling zig-zag movement of the navigator towards a certain direction. The space that is filled can be rectangular with a specified width (a width of 1 gives standard navigational seeking), or

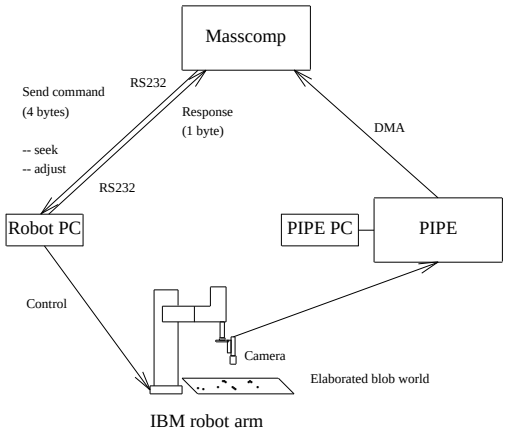

Figure 2: Configuration of the experimental equipment

4 Implementation

The world, map-maker, and navigator were implemented. A Masscomp [15] [16] hosted the map-maker, and the high level control of the navigator. Basic image processing was performed using a PIPE (Pipelined Image Processing Engine) system [3]. The navigator was a camera with a 12.5mm lens mounted on an IBM 7575 robot arm [10], programmed using IBM's AML/2 [11] (Figure 2).

The logical components of our system consists of three main modules. The first, an environment creator, has a user-friendly interface for entering object descriptions and object positions. The second, the navigator module, reads the set of commands generated by the map-maker program and performs the navigation. During the seek phase, the leading edge of the image is monitored by the PIPE. Upon detection, it adjusts and attempts to verify object identity by its dot signature. Some calibration was necessary to ensure that during the monitoring movement was not fast enough to "lose" a dot.

The third component, the map-maker, is the most complex. Much of its efficiency came from the exploitation of the observation that unit directions could only travel within rows or within columns of the world. Being in the same row/column is a necessary but not sufficient condition for an object to be reachable; it also could not be "shadowed" by an object that had an identical leading edge. Note, however, that reachability is not symmetric.

3.4 Definitions of custom map quality

Since there are typically multiple custom maps to travel between any two landmarks in the grid world, it is necessary to define a quantitative model for evaluating the quality of a custom map. There are a number of metrics to measure the cost of a path. Each metric may lead to a distinct path, so each metric may lead to its own heuristics. The choice of metrics depends on the goals of the map-maker and the navigator, and on the characteristics of the grid world. Two possible goals are minimizing travel time and maximizing the probability of reaching the goal.

One obvious cost metric is distance, measured by the Manhattan metric. A second is sensor cost. In our model, this is the number of times along the path that a dot is detected, which reflects the number of times the navigator needs to stop and check if it has reached the next specified object in its set of instructions. In general, the robot may have different sensors with different costs, or the cost to verify an object may vary depending on the landmark.

Time of travel is a third possible metric. Relevant data would include speed limits, obstructions, and sensing times for the available landmarks. Speed limits are generally dictated by law or by roughness of the terrain. The average waiting time for certain obstructions, like traffic lights, may vary depending on the direction of travel. A fourth metric is map length, which is the number of directions that the map-maker gives to the navigator to get from one point to another. This metric measures map transmission costs and memory requirements. A fifth gives preference to straight paths; Elliott and Lesk [6] noted that the number of turns and the direction of the turn also affect the time of travel, and therefore counted right turns as 1/8 mile and left turns as 1/4 mile.

All of these metrics tradeoff with reliability, which generally decreases with distance traveled. This suggests a class of non-monotonic reliability metrics, which would penalize both very short and very long unit direction distances. A second class counts as its cost some function of the confusability of the objects along or nearby the specificed path: some objects are not only unique, they are unique in robust ways. A third class gives preference to "rear view mirror" navigation, that is, prefers landmarks that are distinctive on both sides, so that landmarks can be verified once obtained.

Of course, all of these metrics can be combined in numerous ways. And some or all of these metrics may cause the navigator to back up. For example, suppose the navigator is due west of the destination, all landmarks between the navigator and the destination, including the destination landmark, are red, and a green landmark is found due east of the destination. It is easier to specify the instructions to the destination as "go east until you reach the green landmark, then go west to the red landmark."

3.2 Navigator model

We used a CCD array camera mounted vertically on a robot arm as the navigator's means of observing its environment. The height of camera was fixed above the world model and it could only move in a plane parallel to it. This obviated collision detection algorithms. Because viewing was normal to the ground plane and fixed, there were no occlusions nor even choices of field of view. Field of view was a fixed window, and very small relative to the navigational environment. At most two landmarks were visible to the navigator at any instant; nearly always there was a fraction of one or less.

The navigator is capable of operating in two different phases: *seek phase*, and *adjust phase*. It is assumed to have no metric capabilities outside its view window, and within its view window it perceives by a type of symmetry calculation. During seeking, the navigator continuously looks for any object coming into its viewing window. When an object is spotted on the boundary of the window, the navigator adjusts its position so that the object is centered in the view window. Adjustment not only counteracts errorful drift; it may be strictly necessary on turns to properly align the next heading. Other than seeking and adjusting, the navigator is not intelligent and makes no decisions; the map-maker has made them all. (The opposite extreme is that of a navigator with no *map* at all: everything depends on the intelligence of the navigator's search.) Navigation succeeds if the navigator attains the final object description.

3.3 Map-maker model

In the level helicopter world, each direction in the custom map has the format (*heading, description*). Headings are the four compass points, and the partial object description is simply the leading edge of the 3-by-3 matrix of dots, in the direction of travel. For example, (*north*, '...'), (*west*, ' .') is interpreted as travel north until you see three dots in a row, then move travel west until you see a single dot on the right (i.e., the north). There does not appear to be any advantage in encoding the description further, either in the custom map, or in any of the intermediate data structures used by the map-maker. The object itself, though impoverished, has four different views of three tokens each, and they are trivial to obtain; the object itself has much less. Depending on the ease with which characteristic views can be obtained from objects in general, it may happen that the best description of an object is the object itself. The computational cost of obtaining and storing all views, most of which by definition will not be visible or useful in navigation-in-the-large would be even more excessive than in the special case here.

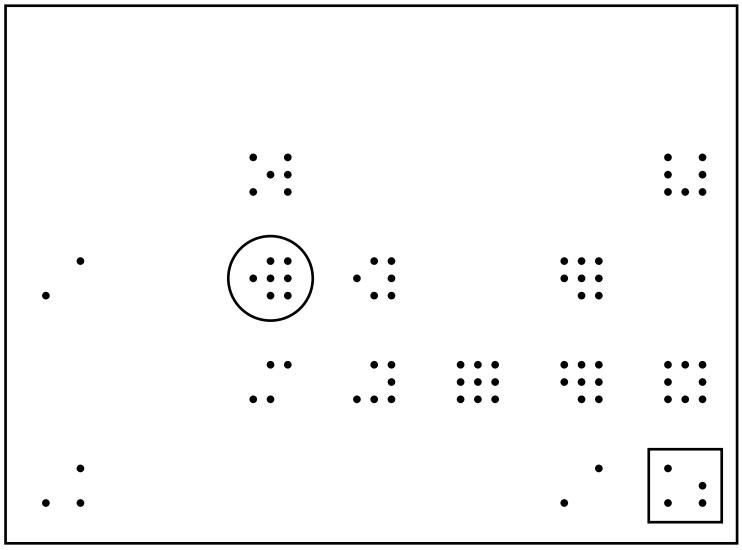

Figure 1: Sample dots world

like mail delivery in offices. Restricting the world to rows and columns still leaves room for a larger vocubulary of heading strategies: various space-filling searches, for example.

An object, or interchangeably a landmark, can be placed in any cell of this grid. Given current limits on processing speed, we define as our object model a 3×3 matrix of dots; this allows 511 possible objects, but far fewer if rotation of objects are not considered distinct. There are no object symmetries due to rotations in this world, since the camera is room-oriented, not path-oriented: that is, as our robot arm changes direction, the camera does not rotate. (This is roughly equivalent to a human being's navigating by turning his road map on his lap as his car turns.) However, all objects must have at least one dot on each of the four edges of the matrix. This simplifies the *adjust* phase of the navigation, and avoids the epistemological issues such as how many objects there are that have a single matrix dot (one or nine?). Figure 1 shows an example of the map-maker's view of our experimental world of size 7×5. The landmark enclosed by the square is the start point and the landmark enclosed by the circle is the goal.

Because there are no occlusions, objects appear the same from all viewing directions; this simplies custom map creation. In general, however, object models must address the characteristic view problem, the occlusion problem, and simultaneously be flexible enough to accommodate multiple kinds of descriptive features that the sensors of the navigator can identify: size, color, texture, shape, number of components, topological relations between components, etc.

	gs	rs	gl	rl	gs	rl	rs	rs	gl
gs		$+r$	$+g$	$+rl$	$+gs$				
rs	$-g$		$+g$	$+r$	$+s$		$+rs$		
gl	$-g$	$-r$		$+r$	$+g$		$+rs$		$+gl$
rl	$-gs$	$-r$	$-g$		$+g$	$+r$	$+rs$		$+gl$
gs	$-gs$	$-s$	$-g$	$-r$		$+r$	$+s$		$+g$
rl		$-rs$	$-gl$	$-r$	$-g$		$+r$		$+g$
rs		$-rs$	$-gl$		$-g$	$-r$		$+r$	$+g$
rs			$-gl$		$-g$	$-l$	$-r$		$+g$
gl			$-gl$		$-g$	$-l$		$-r$	

Table 2: Second stage transition matrix

	gs	rs	gl	rl	gs	rl	rs	rs	gl
gs		1	2	5	6				
rs	1		1	2	3		7		
gl	2	1		1	2		6		8
rl	5	2	1		1	2	5		7
gs	6	3	2	1		1	2		4
rl		6	5	2	1		1		3
rs		7	6		2	1		1	2
rs			7		3	2	1		1
gl			8		4	3		1	

Table 3: Cost matrix

Second Stage In the second stage of transition and cost graph construction, the transition graph is optimized by selecting the minimal object description that maintains object uniqueness. The cost of this optimal unit direction is recorded in the cost graph; sentinel values in the transition graph giving rise to infinities in the cost graph.

In general, the second stage is more complex, conceptually and computationally, for several reasons. First, there may be more than one optimal partial object model that attains the optimal cost: in the example, the unit direction from 1 to 3 can be $+g$, or $+l$. Depending on the overall model of map goodness they may all have to be recorded. Secondly, depending on regularities in the world, or the vocabularies of the navigator directions, the optimization of the entire graph may probably be done in a more efficient way than by optimizing each entry separately. But third and most importantly, the optimization of even a single unit transition is NP-complete in the number of partial models. Selecting the optimal partial model can be obtained by heuristic search, with the most effective heuristic appearing to be the one that says "select the partial model that conflicts most with all the other objects along the path to node j"[13]. In short, the topological visual navigation problem subsumes the problem of minimal object description, except that it provides a well-defined context: the context of the path to the object.

Applying optimization to the example, we find the following full second stage transition and cost graphs, in Table 2 and Table 3, where cost is given as sensing cost over the intervening objects.

2.5 Custom Map Creation

At this point, the problem is no longer a vision or a robotics one; standard graph search algorithms such as Dijkstra or A^* over the transition and cost graph can be used to find the least costly path from a given node to another. The resulting sequence of unit directions is the custom map.

3 Application to the Level Helicopter World

3.1 The World Model

The initial application of these ideas is to an highly idealized two-dimensional world. Nevertheless, it serves as an arena for the exploration of significant computational and representational problems. The world is divided into M rows and N columns, and the navigator's vocabulary of headings is is restricted to traveling in one of the four directions: *north, east, south*, and *west*. Although severely restricted, this world still adequately models much navigation in urban areas, like taxi-cabs in New York, or within man-made structures,

and it may be non-trivial to select the unit directions between nodes in an efficient manner. In special cases, such as the one-dimensional world given above, or in the cartesian two-dimensional world actual implemented, clever encodings do yield good efficiencies of both processing and storage.

For the example, the full first stage transition graph is shown is Table 1, where empty entries are considered to be filled with sentinel values.

	gs	rs	gl	rl	gs	rl	rs	rs	gl
gs		$+rs$	$+gl$	$+rl$	$+gs$				
rs	$-gs$		$+gl$	$+rl$	$+gs$		$+rs$		
gl	$-gs$	$-rs$		$+rl$	$+gs$		$+rs$		$+gl$
rl	$-gs$	$-rs$	$-gl$		$+gs$	$+rl$	$+rs$		$+gl$
gs	$-gs$	$-rs$	$-gl$	$-rl$		$+rl$	$+rs$		$+gl$
rl		$-rs$	$-gl$	$-rl$	$-gs$		$+rs$		$+gl$
rs		$-rs$	$-gl$		$-gs$	$-rl$		$+rs$	$+gl$
rs			$-gl$		$-gs$	$-rl$	$-rs$		$+gl$
gl			$-gl$		$-gs$	$-rl$		$-rs$	

Table 1: First stage transition matrix

In the most general case, the transition graph will be indexed by node number, and the entry for a unit transition between node i and node j will record the heading in its vocabulary and the full object description of j. However, what is most significant is that in general, if navigation is in-the-large, by definition there will be a large number of objects with the same object descriptions; since only the first (that is, the nearest) object is attainable by a given unit direction, the transition graph will always be sparse, and it may be compressible by the usual techniques.

We now note that one definition of a "landmark" would be an object whose column in the transition graph is very nearly filled, that is, an object that can be obtained under a unit direction from very nearly everywhere. In the above example, the gl object is an optimal landmark. One can easily define a concept such as the "landmark radius" of an object in the obvious way. Because landmarks can be used to navigate over great numbers of objects, they will tend to appear regularly in custom maps. However, landmarks need not be "reversible"; for example, the fourth object in the above example is the first of two such objects close together, and is only valuable as a landmark while navigating to the right; it is perceptually "shadowed" by the sixth object when navigating to the left. In other words, traveling left from most places it is impossible to reach it in one step.

tunately, we first need some additional definitions and representations to investigate how difficult these questions are; we present them in the context of the one-dimensional world example. Their extension to two-dimensional space is conceptually straightforward, but computationally daunting.

2.4 Transition and Cost Graphs

We introduce two abstract data types, the transition graph and the cost graph. They are abstract data types because, depending on the vocabularies for headings and object models, the map criteria being optimized and, in some cases, the contents of the world model itself, they are realized as various data structures. Thus, in some cases they are best implemented as arrays, in others as lists or trees. In some cases they are best established before the optimizing map-making algorithm, in others they are incrementally established cooperatively with it, and in still others they are never established at all, but are directly incorporated into the algorithm itself.

Intuitively, the graphs record information about the most efficient direction (heading plus object description), if any, that takes a navigator from any given node in the world model graph to any other node. If there is such a unit direction, the transition graph records it, and the cost graph records its cost. This cost can be defined in multiple ways: it can be the distance traveled, or the amount of sensing necessary to tell that intervening objects are not the goal, or simply the complexity of the heading strategy. Both graphs conceptually contain a special sentinel wherever a unit transition is impossible, which is usually the case; in particular, there is no unit direction that can cause the navigator to remain in the same place.

The problem of obtaining an efficient custom map from a start to a goal can then be broken down - conceptually, at least - into the establishment of efficient unit directions, followed by the traversal of the unit direction graph from start to goal. There are times, of course, when this is grossly inefficient, such as when the navigator is traveling a very short distance and a type of search is more appropriate; we do not thoroughly analyze such tradeoffs here.

Intuitively, the graphs are filled in in two stages.

First Stage; Formal Definition of "Landmark" In the first stage the transition graph is filled with tentative unit directions of the most inefficient kind, namely the full object description of the unit direction's goal. For example, in the example one-dimensional world model (gs)–(rs)–(gl)–..., the unit transition connecting nodes 1 and 3 would initially be given as +gl, even though either $+g$ or $+l$ would suffice. In the most general case in two dimensions, creating the world planar graph, and labeling the arcs between two reachable nodes is a complex procedure, highly dependent on the vocabulary of headings and objects,

and both are ignored in our world. The first purpose is, they calibrate the navigator during an interactive direction-giving (for instance, the navigator can ask "*What's a W?*") but our direction-giving is not interactive. Secondly, they serve as a warning that a neighborhood is becoming visually confusing, and that more precise sensing is called for, (effectively, Scan for X-confusable objects); our navigator has no sensory choices.

2.3 Navigating with a Custom Map

Having defined what a custom map is, we now examine its use. Then we will address its creation, which is one of the central concerns of this paper.

The most primitive decision a navigator must make is whether or not its observation of the world is compatible with some object description given by the map-maker. We model such an operation by a boolean-valued function "match", which takes as input a description, and uses the description to schedule the appropriate sensory and cognitive information to make the judgment.

To illustrate navigation, consider the following world model, which is impoverished to the point of being linear. Objects are described by two point-like features, color and size, taken from the domains red, green and small, large, respectively. There is no spatial relations in the world other than forward and backward ("+"/"−"), and there is no intra-object spatial relations at all. The world is given by the graph (gs)–(rs)–(gl)–(rl)–(gs)–(rl)–(rs)–(rs)–(gl), which is a straightforward syntactic simplification of the full definition of (*green, small*)–(*red,small*)–. . . Directions such as $+rl$, or $-gs$, or even $+l$ tell the navigator to move forward until the red large object is sensed, or backward to the green small one, or forward to the large one, respectively. The last direction is a partial object description, and the ability to partially describe landmarks greatly complicates the selection of landmarks and their descriptions, although it likewise greatly simplifies navigation itself.

The navigator requests a custom map to traverse from the first object to the next to last one. The epistemological issues of how a navigator knows and/or communicates both present and desired positions to the map-maker are side-stepped here. The custom map maker can reply with over 300 custom maps; among them are $+s+s+s+s, +g+gl-r, +gs+s+r, +lr+sr+r, +g+r+g+r+g+r+r+r$. The first map uses only one object model. The second causes overshooting of the goal, but has only three directions, and therefore minimizes communication time. The last map is one of many that minimizes distance, but enumerates all objects between the start and the goal. Depending on the way in which sensory costs are tallied, the third or fourth may minimize sensing. Even in one-dimension, there is great variety possible.

In two dimensions, given a vocabulary of headings, and a vocabulary of object models and their allowable partial models, the number of potential custom maps grows exponentially. Thus, we would like to make the idea of "efficient" and "best" map precise. Unfor-

2.2 Landmarks and Custom Maps

To consider objects as point-like requires enough processing intelligence in the navigator to recognize an object as a single object, and to capture, hold, and dismiss the single object from its sensory array; in effect, the navigator can "debounce". This is an important consideration, since the "experience" of an object must be a unified whole, even if the prior and succeeding experiences are identical. Practically speaking, this limits the amount of blind travel between objects; further, it defines what is meant by a single object from the point of view of the navigator. Three identical trees as seen from above must give rise to the experience of "tree" exactly three times, even though the trees are sensed at some distance, have spatial extent, and may overlap, and are still sensible after some distance, often even when the next tree is also sensible.

Given the model of unified, immediate perception, which implies that interframe time is small, objects can now be modeled by the map-maker by some symbolic abstraction. This object model must be communicable to the navigator, and the navigator much be capable of modeling its own perception in these terms. It is clear that the grammars for describing objects are many, but they are assumed here; this paper is more concerned with inter-object distinctions and relationships. The definition of a landmark ultimately invokes the ability of a navigator to distinguish a landmark from any other object in its graph neighborhood; a landmark is an object that can be recognized by a navigator along a path through the world graph, regardless of the intervening object nodes. Thus a tree in a desert is landmark, but a tree in a forest is not. Landmarks can be relative, or even "one-sided" - that is, path-dependent; for example, a tree on a forest's edge is a landmark only on the way in. We will make the definition of landmark more precise shortly, at least for our level helicopter world.

A custom map is a sequence of directions; each direction is a pair consisting of a heading and a landmark description. The navigational goal is simply the last landmark description. The first item of the pair, the heading, can be quite general, as it can specify a strategy (*wander northward, spiral outward*), as well as an absolute heading (*east*) or relative heading (*turn left*). In extreme cases, the navigable regions between landmarks are capable of being continuously sensed as a series of landmarks, e.g., a road and its sides, and the heading can be specified in terms not addressed in this paper , e.g., follow the road. The second item of the pair, the landmark description, can be arbitrarily complex, not only with respect to the need to efficiently transmit an object model, but also with respect to any spatial grammar superimposed on object primitives, e.g., "three occurrences of big houses next to things other than a big tree or a small pole". The theory of LR(k) parsers addresses some of these issues, since basically a direction based on a compound object is a simple sensory program.

We note as a side issue that our world ignores those "reassurance" directives that are common in human map-making, e.g., "along the way you will pass a W", or, "if you see an X you have gone too far, turn around and look for the Y." Their purpose seems to be twofold,

We find it critical to distinguish three similar but subtly different perceptions of "the world".

The two-dimensional world as it exists and is experienced by both map-maker and navigator 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: many such worlds consist largely of the essentially empty space between objects, much navigation can be done by simple object order rather than object distance, and objects can often be considered to be point-like.

We therefore postulate that the map-maker, omniscient and error-free, has abstracted the world into something more akin to a planar graph. That is, the world is conceived as a collection of nodes which represent significant objects, connected by arcs which represent navigable regions without intervening objects. Empty space, distance, and object extent are ignored. The omniscience of the map-maker ignores the very difficult issues of map induction (see [5]), and the infallibility dodges the issues of partial or errorful information, deliberate camouflage, unexpected events, or even the proper navigator strategies for "believing" a map and for recovering from any errors. However, to our map-maker, a plane flight would be simply the graph (or subgraph), $(New\ York)$–$(London)$–$(Paris)$–$(Rome)$.

The map-maker communicates even less of this subgraph to the navigator in the form of a sequence: this is, the best, by some criterion, "custom map"of visual landmarks, ordered by sequence of traversal but not ordered by space, or even by actual time of encounter. In fact, the actual navigation path can be self-intersecting; in some cases, in order to be optimal, it must be. It is this second level of abstraction and the process of its creation are the foci of the paper. Throughout we assume that there is only one navigator, and that the map-maker's omniscience extends to a perfect model of the navigator's sensory endowment.

The navigator perceives the world in the most limited way. Sensory range is limited, perhaps only to the current object being experienced, and there is virtually no memory. However, this is an actual advantage; usually the less that needs to be sensed or remembered the better, since the cognitive load of the navigator is minimized. Although more intelligent navigators can be modeled, this one chooses to ignore most of the world and its features for the sake of efficient traversal.

For consistency, we will refer to the abstraction of the world as the "world model" or the "planar graph", and the custom map as, simply, the "map". In layman's terms, the world model is itself usually called a "map", particularly if it is not drawn to scale, and the custom map would probably be called "the list of directions." However, in some instances, particularly at car rental agencies equipped with electronic custom map-makers, the layman can be given exactly what we call a map, if he has a unique destination in mind. Surprisingly, recent research shows that maps as we define them are preferred by human beings over the Exxon-variety world models [18].

1 Introduction

In this paper, we define a model and describe an implementation for characterizing a class of visual navigation problems in-the-large. Although both model and implementation are extendible, this paper focuses on visual landmark selection and topological navigation within (actually, above) a two-dimensional environment, without the use of metric distance information, and without the use of "graphic" information such as street signs. The domain we explore is a formalized variant of the visual world seen by a level-flying helicopter whose perception is limited to a narrow field of view, and whose visual axis is normal to the world below it. Thus, occlusions are not yet an issue, but object identity and confusability are, and so are issues of communicative, perceptive, and motive economy.

This work is ultimately motivated by the questions of what is a good map, what is a good landmark, and what is a good visual environment. It addresses the abstract problem of how to optimally choose visual landmarks and their sensory features in order to describe and discriminate objects, and how to create short or efficient sequences of such descriptions for low cost (however defined) qualitative navigation from a given place to another.

2 Formal Definitions

2.1 The World, The Map-Maker, and The Navigator

The world we model is two-dimensional in the sense that objects are considered to be equivalent to their appearance, from a fixed vertical angle, and within a narrow, and therefore orthographic, field of view. In that sense, it is very similar to the world as it would be displayed in an aerial photograph, if that photograph has no symbolic information superimposed upon it. Since the photograph has not been taken at an oblique angle to the ground plane, there are no significant occlusions. We further restrict the navigator's movement to be in a constant plane above and parallel to the ground plane, and the navigator's vision to be normal to it.

Relaxing the restriction of constant plane would complicates the giving of directions, since viewing height would be variable, would have to be specified by the direction-giver, and introduces an additional degree of freedom over which the choice of landmarks would have to be optimized since most landmarks are only landmarks relative to their visible surround. Relaxing the restriction of normal view would be more critical, since it introduces two degrees of freedom and introduces visual occlusions. In principle, the possibility of occlusion deeply impacts the definition and detection of landmarks, since at least part of a landmark's effectiveness comes from its visibility over long ranges. We restrict ourselves now, however, to visual navigation with two degrees of freedom over a two-dimensional world.

A Formalization and Implementation of Topological Visual Navigation in Two Dimensions

John R. Kender
Il-Pyung Park
David Yang
Department of Computer Science
Columbia University
New York, NY 10027
CUCS-021-90

April 30, 1990

Abstract

In this paper we formalize and implement a model of topological visual navigation in two-dimensional spaces. Unlike much of traditional quantitative visual navigation, the emphasis throughout is on the methods and the efficiency of qualitative visual descriptions of objects and environments, and on the methods and the efficiency of direction-giving by means of visual landmarks. We formalize three domains–the world itself, the map-maker's view of it, and the navigator's experience of it–and the concepts of custom maps and landmarks. We specify, for a simplified navigator (the "level helicopter") the several ways in which visual landmarks can be chosen, depending on which of several costs (sensor, distance, or communication) should be minimized. We show that paths minimizing one measure can make others arbitrarily complex; the algorithm for selecting the path is based on a form of Dijkstra's algorithm, and therefore automatically generates intelligent navigator overshooting and backtracking. We implement, using an arm-held camera, such a navigator, and detail its basic seek-and-adjust behaviors as it follows visual highways (or departs from them) to reach a goal. Seeking is based on topology, and adjusting is based on symmetry; there are essentially no quantitative measures. We describe under what circumstances its environment is visually difficult and perceptively shadowed, and describe how errors in path-following impact landmark selection. Since visual landmark selection and direction-giving are in general NP-complete, and rely on the nearly intractable concept of characteristic views, we suggest some heuristics; one is that the landmark object "itself", rather than its views, may be its most compact encoding. We conclude with speculations about the feasibility of intelligent navigation in very large self-occluding visual worlds.