Error Detection and Recovery in Two Dimensional Topological Navigation

Il-Pyung Park and John R. Kender
Department of Computer Science
Columbia University
New York, NY 10027
{ip, jrk}@cs.columbia.edu
CUCS-002-94

Abstract

In this paper we describe error detection and error recovery methods applicable to navigation in large scale unstructured environmental navigation. We relax the assumption of error-free following of topological landmarks; the navigator is "permitted" to make mistakes during its journey. The error detection method involves the navigator observing its immediate surrounding and checking for one of several types of disparities. The error recovery method is based on a simple fixed set of movements which is triggered by the navigator's local observation. Alternately described, this work characterizes those environments in which robust topological navigation is possible, including those landmarks which, literally, "can't be missed". These methods have been implemented on our qualitative environmental navigation system consisting of a camera mounted IBM 7575 robot arm.

Relevant Categories: Qualitative vision, Vision-guided robotics

1 Introduction

Our navigational environment consists of identically featured objects scattered randomly on a flat surface [PK93b] [PK93a]. This simplification of the world emphasizes the spatial and topological problems of doing vision and navigation in a large and unstructured environment. That is, an object cannot be described by its intrinsic attributes, such as shape, size, or color. Therefore, in order to describe an object (and also to recognize it), the geometrical relationships of the target object to its neighboring objects must be considered. The directions of the movements of the navigator have no external reference to rely on, such as roads or compasses.

In a more complicated world that has a more variety of objects, we can (and often humans do) still apply the same abstraction. For example, in the left diagram of figure 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 location in the environment. This is shown on the right diagram of figure 1. Another way of abstracting the world would be to categorize the objects according to their types (using whatever categorizational strategy is available to the navigator), and produce n (number of object types) separate dots world. This is shown in figure 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.

1.1 The map-maker and the navigator

Here, we briefly describe a navigational system for a such environment as presented in [PK93b]. The purpose of the map-maker is to generate a custom map, consisting of directional instructions, so that the navigator will be able to find its way to the destination by executing each of the

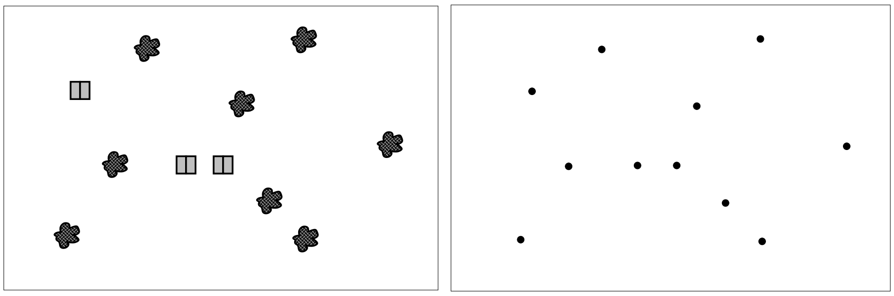

Figure 1: A world consisting of trees and houses, and the corresponding dots world representation

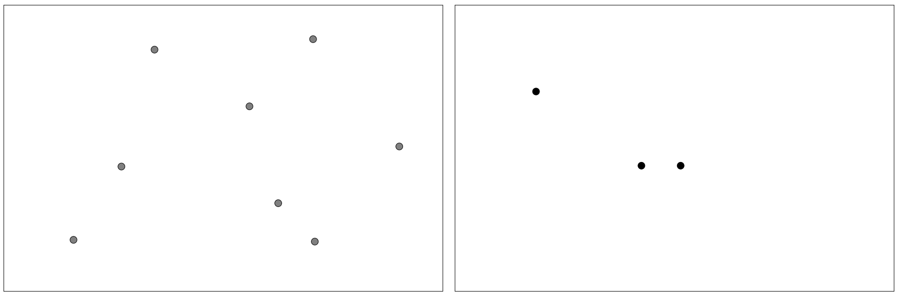

Figure 2: A separation of the dots world according the the navigator's sensibility: only trees, or only houses

instructions in a sequence. The map-maker is assumed to be omniscient and error free (although these assumptions are relaxed in a later stage[PK93b][PK93a]), and knows the capabilities and the limits of the navigator. Therefore, the custom map contains directional instructions that the navigator can handle. The communication between the map-maker and the navigator is done off-line (one-way, one-time). This means that the custom map is given to the navigator in the beginning of the journey and the navigator's only source of directional information is the custom map.

The navigator's capabilities are much more limited. Its view window size is very small relative to the environment. It has limited metric measurement capabilities only within its current view window. Therefore, it is not possible for the map-maker to give, to the navigator, the (x, y) coordinate of a landmark as part of the directional instructions. The directional vocabulary in the custom map must be qualitative. This simple, almost dumb, navigator model is justified by our purposes in this project: (1) Efficiency of navigation (2) Low cost navigation (3) Finding the lower bounds of navigator's intelligence without sacrificing the performance (4) Humans seem to prefer this model[SVW85].

1.2 Parkways and Trajectories

Prior to the navigation, the map-maker captures the entire world by taking the image of the world from a global location. The positions of the populated objects are recorded. The map-maker further abstracts the world into a directed graph data structure, with vertices and edges. We define that object b is connected to object a, if a is at a reference position (one of 4 corners of the view window) and b can be identified as a unique landmark by a qualitative description vocabulary (isolated, obvious, etc.) [Par93]. By applying the connected component algorithm using the above definition of "connectedness", we can then generate discrete sets of connected components. We define parkway to be such a connected component. For the purpose of this paper, we concentrate on a specific type of parkway called "isolated parkway, which consists of isolated landmarks [PK93b]. Traveling within a single parkway is then simply following visible landmarks. This guarantees that the robot does not move via metric information outside its sensible region, thus eliminating the possibility of a dead reckoning error.

To traverse between two disjoint parkways, we use the "trajectory" method, in which the navigator "slides" the view window in the direction formed by two objects within its view until a new object is reached. The inter-parkway paths generated by this method are defined as trajectories. Note that we are not pinpointing any coordinates for the navigator's movement, but in fact the navigator's movement is based on an orientation formed by landmarks. Figure 3 shows parkways

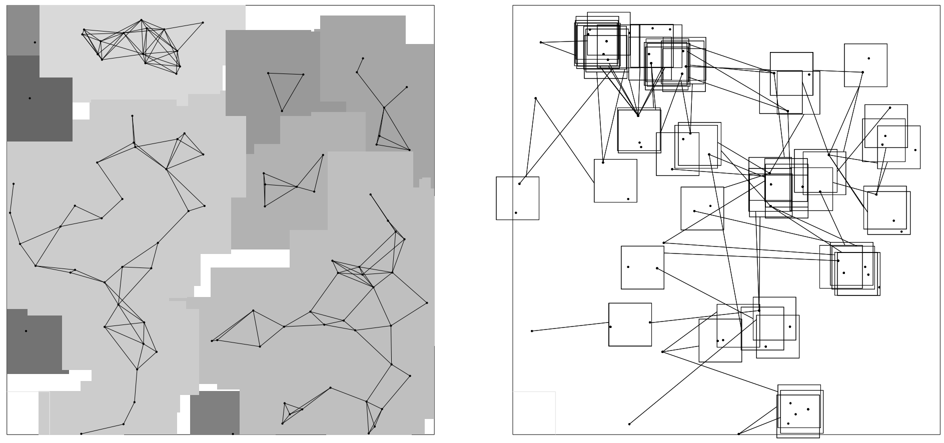

Figure 3: Parkways for a randomly populated world with 100 objects (left), and the corresponding trajectories (right)

formed on a world of 100 objects (left diagram), and the corresponding trajectories (right.) The navigator's window size is 10×10, and the world is 100×100.

The overall path consists of parkway paths and trajectory paths. The map-maker further abstracts the computed path into a list of directional instructions, called custom map, which is used by the navigator during its physical encounter of the real world. Each entry in the custom map is of the form $(D, [T], M)$, where D is a description vocabulary (such as the isolated point descriptor), $[T]$ is an optional term that indicates trajectory, and M is the corner designator that indicate how to move after identifying a landmark. The map-maker and the navigator was implemented using an IBM 7575 SCARA arm, two CCD cameras, PIPE (Pipelined Image Processing Engine), and a Sun workstation with X-window interface. Figure 4 shows the configuration of the map-maker and the navigator. Figure 5 shows the map-maker interface program displaying a generated path (right to left), with corresponding custom map:*((isol, SW), (isol, SE), (isol, SE), (isol, NE), (isol, trajp, NE), (isol, NW), (isol, NE), (isol, NE), (isol, *)).* Details, including sources and sensitivity

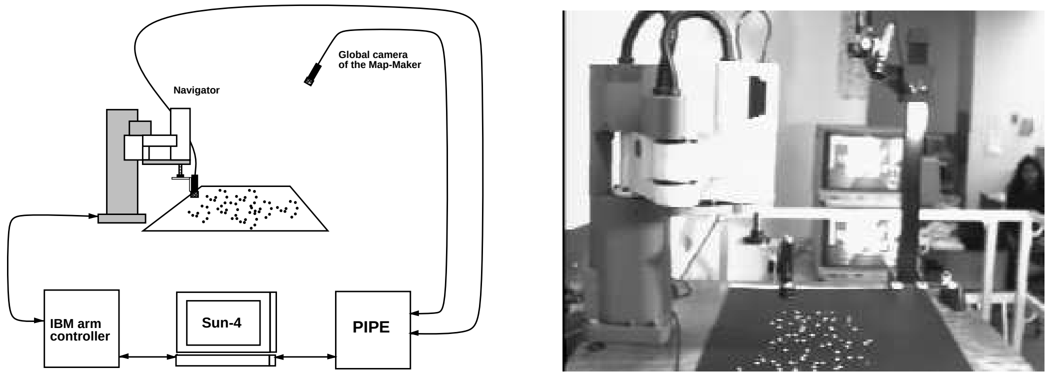

Figure 4: The experimental setup configuration

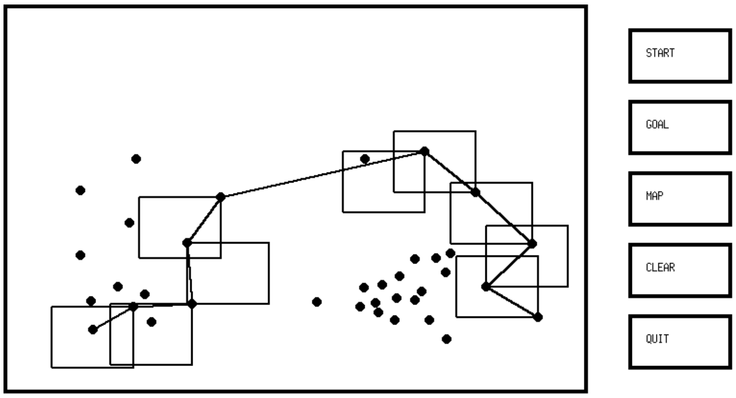

Figure 5: Interface program showing generated path

to errors, can be seen in [PK93b][PK93a].

In this paper, we relax the error-free assumption of the map-maker. The navigation can therefore fail due to either the errors made by the map-maker or the navigator. We focus on the issues relating to the detecting and recovering from the mistakes made by the navigator. There are two different ways to deal with errors. The first method is "error prevention" and is performed by the map-maker. Prior to the navigational task, the map-maker carefully examines the environment and generates a path that is least affected by error, thereby preventing the navigator from making mistakes. This has been described, analyzed, tested, and measured experimentally in [Par93], and in some cases, closed form results have been obtained. The second method is "error recovery", and is accomplished by the navigator during its navigational task. At each "check-point", which is a landmark, the navigator examines its immediate surrounding to verify that it is on the correct course. If an error has been detected, it carries out an error recovery routine to get back to the original course[1]. In this paper, we describe the latter.

2 Error detection

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

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

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

where Observe(Q) is a description of the corresponding quadrant $Q \in$ {SE, SW, NE, NW}. In the isolated landmark selection process, a wrong object may be mistakenly chosen. If the signature of the wrongly chosen object is different from the signature of the correct isolated landmark object, then the mistake can be identified by comparing the desired signature to the currently observed

[1]Our error recovery method can be classified as "forward recovery", as defined in [Stu92], because it replans to get back on track, instead of retracing its traveled route.

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, as listed as follows:

1. Quadrant Occupation Count (QOC) - is defined to be a sum of 4 numbers (0 or 1) indicating the presence or the absence of objects in each quadrant. i.e., QOC $\in \{0, 1, 2, 3, 4\}$. QOC can be represented by a 2-bit number. In figure 6, the QOC of object R is 3 because 3 of the quadrants centered at R are occupied.

2. Quadrant Occupation Signature (QOS) - is defined to be a concatenation of 4 numbers (0 or 1) indicating the presence or the absence of objects in each quadrant. i.e., QOS $\in \{0000, 0001, \ldots, 1111\}$. QOS can be represented by a 4-bit binary number. In figure 6, the QOS of object R is 1110.

3. Quadrant Population Count (QPC) - is defined to be a sum of 4 numbers corresponding to the number of visible objects in each quadrant. i.e., QPC $\in \{x|$ integer$\}$. QPC can be represented by an integer. In figure 6, the QPC of object R is 9.

4. Quadrant Population Signature (QPS) - is defined to be a concatenation of 4 numbers corresponding to the number of visible objects in each quadrant. i.e., QPS $\in \{x_1x_2x_3x_4|x_i$ is an integer for all $i\}$. QPS can be represented by a 4 column valued vectors. In figure 6, the QPS of object R is 5220.

For our implementation and experiments, we used the Quadrant Occupancy Count (QOC). Details on how QOC was used for error detection and recovery are explained in the following

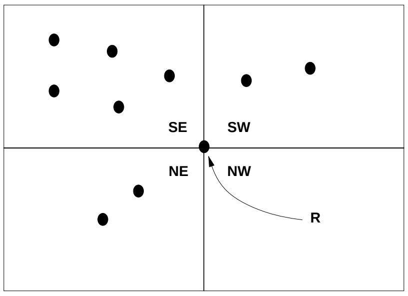

Figure 6: Example configuration for Observe() function

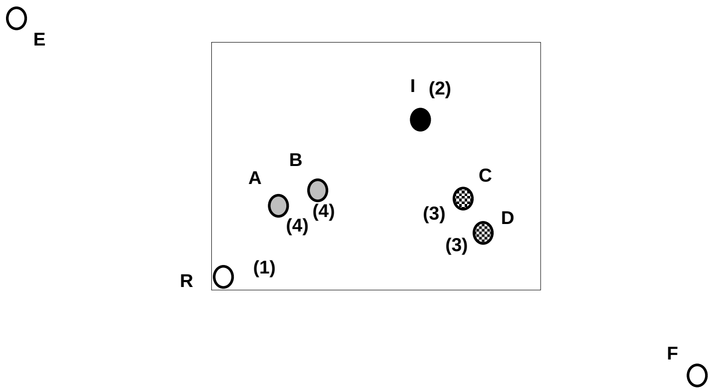

Figure 7: Error detectable configuration

sections.

Error detection using QOC QOC of a landmark is obtained by moving the view window around the 4 quadrants relative to the landmark and by counting the number of occupied quadrants. Errors can be detected only if the landmark has a unique QOC within the current view window. In figure 7, we see there are 5 objects in the view window. R is the current reference point and I is the desired isolated landmark. QOC of the visible objects are shown in table 1. As we can see, QOC of object I is unique within this view window, thus error can be detected by examining the QOC of the

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

Table 1: QOC values of objects A, B, C, D, I, and R

	NW	SE	SW
A	0	1	1
B	1	1	1
I	0	1	0

Table 2: Signature string of objects A, B, and I

obtained landmark.

Optimization of sequencing window positions To obtain the QOC of a landmark, the navigator needs to examine its 4 quadrants (NE, NW, SE, SW). We know that one of the quadrants is always occupied because of the reference point. Therefore, we need only look at 3 quadrants. If we use QOS to identify a landmark, by sequencing the 3 quadrants in a certain order, we can detect for error (or validate the current move) before all 3 quadrants are examined. For example, let us suppose that NE is occupied by the reference point, and therefore we need to examine only NW, SE, and SW. Let a string BBB indicate the occupancy of the corresponding quadrants, where B=(0,1). Let us further assume that there are 4 objects in the window, A, B, R, and I, where R is the reference point and I is the isolated point. Assume that the signature string for A, B, and I are as in table 2. Then, examining NW *alone* will determine whether or not the navigator is in the correct place, instead of examining all 3 quadrants. Unfortunately, choosing the optimal set of quadrants (either in serial, or in parallel), is an NP-Complete problem because this problem can be reduced to the minimal set covering problem. But, since we have only 3 quadrants to work with, exhaustive search is possible.

3 Error recovery

In the preceding section, we described methods to detect errors during navigation. Error detection is useful only if there are corresponding error recovery methods available to the navigator. In this section, we describe a such error recovery routine that the navigator can execute when errorful situations are encountered. In the worst case, when an error is detected, the navigator simply gives up its navigational task and says "I am lost". But in many cases, by performing a fixed set of "reset" sequences, the navigator is able to get back to its originally intended course.

General algorithm The basic idea is to assign a fixed action for each QOC (0 to 4). Therefore, the error recovery is a simple "if QOC then ACTION" set. For example, in figure 7, objects A and B have QOC of 4. If the navigator chooses either A or B instead of the desired I, it can "correct" the mistake by moving to the SW corner (relative to A or B) and then choosing the isolated point. Likewise, the correction instruction for objects C or D whose QOC is 3, is (SE, Isol).

Formally, an isolated point is error recoverable if:

1. The configuration is error detectable.

2. Self containment: Objects in the same QOC group are positioned so that the originally intended landmark (the true isolated landmark, I) is in the same quadrant related to these objects. For example, in our example of figure 7, objects A and B having QOC 4, both have I in quadrant SW.

3. Transitively reachable: 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

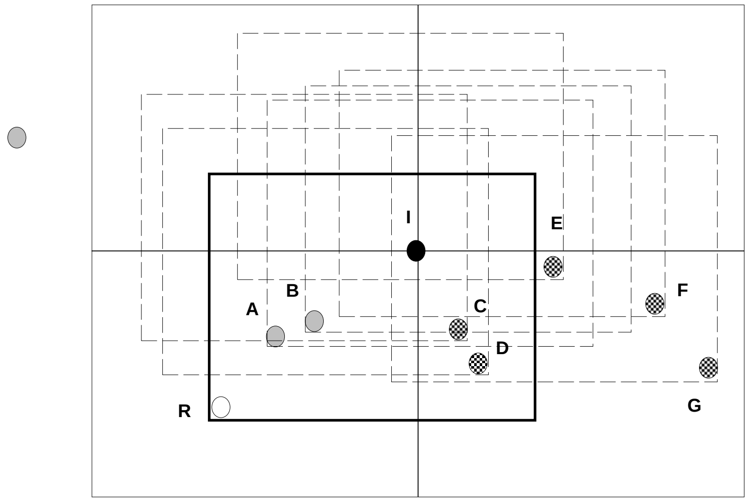

Figure 8: Error recovery domain

our figure 7, the isolated point of object C in SW quadrant is in fact object B, not I. But I is reachable from B. Therefore, I is transitively reachable from C.

Reachability domain Before the error-recoverability conditions, as suggested in previous paragraph, can be tested, we need to find all objects that are reachable - the objects that can be wrongly chosen as the true isolated landmark. For example, in figure 8, every object in the original view (inside box with dark solid lines) are reachable. Object E is also reachable from object A and B, but objects F, and G are not.

Following is a simple algorithm to identify all the objects that are reachable during the error recovery procedure.

1. Consider all objects (except for the isolated point) within the 4 quadrants spanned by the true isolated point (thin solid lines in figure.) With these k objects, form a $k \times k$ matrix. For example, in the figure, we have 7 objects A, B, C, D, E, F, and G.

2. Initialization of the matrix *reach*[i, j]: If the i-th object is within the original window, update

	R	A	B	C	D	E	F	G
R	1	1	1	1	1	0	0	0
A	0	1	1	1	0	1	0	0
B	0	0	1	1	1	0	0	0
C	0	0	1	1	0	0	0	0
D	0	1	1	1	1	0	0	0
E	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0

Table 3: $reach[]$ matrix after running the domain algorithm

$reach[i, i]$ of the matrix as '1'. Everywhere else, put 0.

3. Update matrix: For each object, find out which object it covers (except for itself) and update the matrix accordingly. For example, in the figure, object A covers objects B, C, and E, therefore, $reach[A, B]$, $reach[A, C]$, and $reach[A, E]$ are set to '1'.

4. Reduce the matrix: Examine every column of the matrix to see which column has all-0 entries, and update the corresponding (transpose) row to 0. For example, in the figure, object G has all-0 column. Therefore, every entry in row G of the matrix is set to 0. Repeat step 4 until no further update is possible.

5. Objects that have non-zero columns are reachable.

The $reach[]$ matrix of configuration in figure 8, after running the algorithm, is shown in table 3. As we can see, F and G have 0-columns, therefore are not reachable. This reachability domain problem can be solved by a version of "topological sorting" algorithm and therefore within time complexity $O(|E| + |V|)$ [AHU74].

Figures 9 shows an example of error recoverable parkways formed in a larger domain. Note

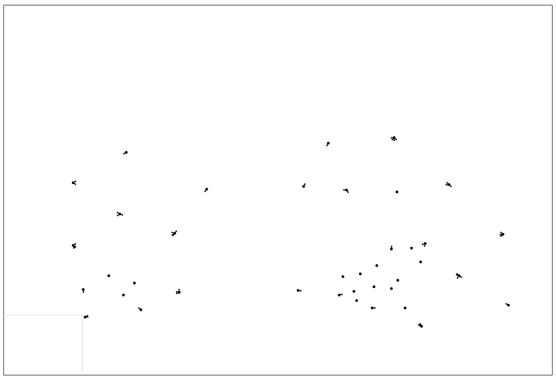

Figure 9: Error recoverable parkways

that parts in congested regions are not on the parkways.

4 Error detection in trajectories

We can use the QOC methods to detect errors in the trajectory movements by appropriately modifying the existing algorithm introduced earlier. In the parkway case, the landmark needed to have a unique QOC within the view window in order for it to be error detectable. In trajectories, the landmark (trajectory target) has to be unique among all the reachable objects. Recall, from previous section, that an object is reachable, during a trajectory movement, if there is a possibility of navigator obtaining it before the trajectory target.

5 Error recovery in trajectories

A trajectory movement is error recoverable if the configuration satisfies the three conditions (similar to the parkway error recoverability):

1. The trajectory movement is error detectable.

2. Self containment: Similar to the parkway case.

3. Transitively reachable: Similar to the parkway case.

Figure 10 shows an example of an error recoverable path. The initial position of the navigator is at the lower left hand corner. Initial reference object is the dark gray object. The goal object is near the top right hand corner, represented by the light gray object. The numbers next to the objects are the corresponding QOC's. Initially, the navigator sees 5 objects in its view. The first entry in the custom map instructs the navigator to identify isolated landmark and adjust to SW. If it makes a mistake and wrongly chooses an object whose QOC is 4, it is told to adjust to SW and look for an isolated landmark. If it chooses an objects whose QOC is 2, it is told to adjust to SE and look for an isolated landmark. Only when the chosen object's QOC is 3, it is able adjust to SW and continue to the next instruction. The next entry in the custom map instructs the navigator to do a trajectory movement based on directional vector formed by the reference point and the isolated point. If the QOC of the obtained landmark is 2, it is told to adjust to NE and to look for isolated landmark. Otherwise, if it is 1, it can adjust to SW and continue to the next instruction. The last entry in the custom map tells the navigator to identify an isolated landmark as the goal object. If the computed isolated point has QOC of 2, it is told to recover by adjusting to NW, and then by identifying the isolated landmark. Otherwise, if the QOC is 3, the goal has been identified.

6 Results

6.1 Experiments

We have implemented the QOC method as described the previous section. In figure 11, we see a path generated in a simple environment. The starting position is at the lower right-hand corner. Corresponding custom map for this path is:

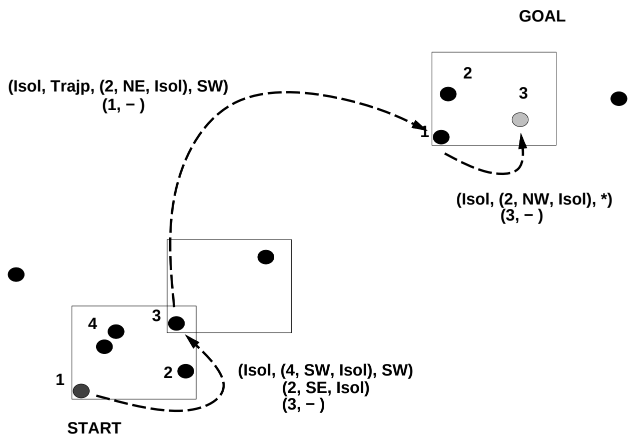

Figure 10: An error recoverable path

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

In our test run with the robot, the navigator failed to 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, QOC was 4, which indicated the robot to adjust to NE and to recapture the correct landmark. In this case, the error recovery routine handled the error caused by the object which was positioned near the boundary of the view window. After the error recovery routine, the navigator was able to get back on track, and eventually get to the destination successfully. 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.

Figure 12 shows a path generated by the map maker that is highly error correctable. The navigator starts near 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

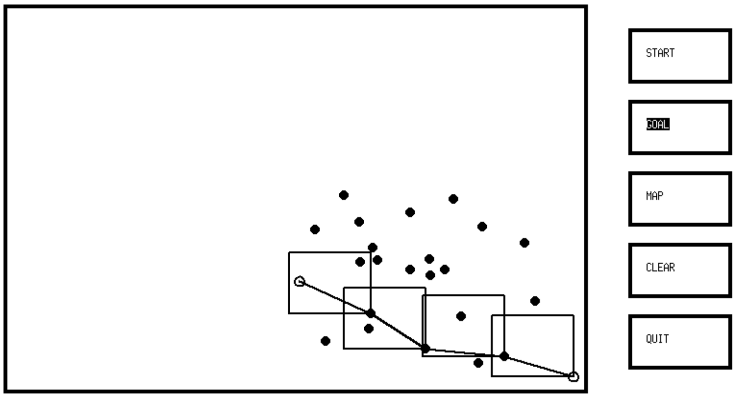

Figure 11: Example of a simple error recoverable path (path starts in the lower right).

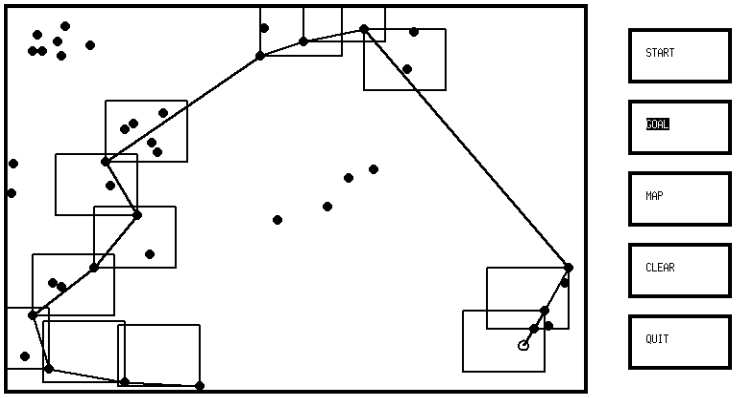

Figure 12: A highly error correctable path (path goes clockwise).

on the user interface display during the map-making session), including error detection / correction statements are shown in figure 13.

6.2 Categorization of success and failure cases

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. We have categorized these as the following.

Success

```
 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
rum>
```

Figure 13: Custom map commands as shown to the user during the map-making session.

1. The navigator follows correct path and reaches the goal directly. The navigator may or may not be equipped with error handler routines. If the navigator is provided with error handlers, all verify correctly during the navigation without having the need to execute recovery sequences. The navigator is aware of the success.

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.

3. The navigator follows wrong path, but ends up correctly. In this case, the navigator makes incorrect moves and/or incorrect validations and/or recoveries, but somehow ends up in the right place without knowing its mistakes. The navigator is aware of the success, but is unaware of its mistakes.

Failure

1. The navigator stops due to inability to proceed, case 1: In this case, the navigator looks for a landmark, but is unable to locate it. For example, the custom map instructs the navigator to identify an obvious point, but there are more than one objects visible (no obvious point). The navigator is totally "lost" and the navigation has failed. The navigator is aware of the failure.

2. The navigator stops due to inability to proceed, case 2: Consider a situation where a landmark is error detectable, but not error recoverable. If correct landmark is verified, the navigator is able to proceed with its confidence reset. 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 not "lost", but simply doesn't know how to correct itself.

3. The navigator follows an incorrect path to its end. In this case, the navigator is unaware of it mistakes (incorrect moves and/or validations), and still thinks it has reached the correct goal. The navigation has failed. The navigator is unaware of the mistakes and is also unaware of the failure.

7 Summary

In this paper, we proposed a general framework for error detection and error recovery. The basic idea is to have the navigator examine its immediate surroundings at predefined check points (landmarks). Where and how many check points there are, depend on the environment. In general, the check point landmark must have some unique feature that distinguishes it from its neighbors. For our implementation of the error detection and recovery, we used Quadrant Occupancy Count (QOC) to characterize the current surroundings of the navigator, which represents the presence or the absence of objects in the neighborhood of the check point landmark. When the QOC of the obtained landmark does not match the anticipated value, the navigator is able to be told to execute a sequence of error correcting movements to get back on the track.

References

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

[GK78] K. Chidananda Gowda and G. Krishna. Agglomerative clustering using the concept of mutual nearest neighborhood. *Pattern Recognition*, 10:pp105–112, 1978.

[KL89] John. R. Kender and Avram. Leff. Why direction-giving is hard: The complexity of linear navigation by landmarks. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6):1656–1658, 1989.

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

[Kui79] B. Kuipers. On representing commonsense knowledge. In *Associative networks*, pages 393–408. Academic Press, 1979.

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

[PK93a] Il-Pyung Park and John R. Kender. Qualitative environmental navigation. In *Proceedings of the Image Understanding Workshop*, 1993.

[PK93b] Il-Pyung Park and John R. Kender. Using isolated landmarks and trajectories in robot navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society Press, 1993.

[Stu92] Elizabeth Stuck. *Detecting and Diagnosing Mistakes in Inexact Vision-based Navigation*. PhD thesis, University of Minnesota, 1992.

[SVW85] L. A. Streeter, D. Vitello, and S. A. Wonsiewica. How to tell people wherer to go: Comparing navigational aids. *International Journal of Man-Machine Studies*, 22:549–562, 1985.