Maintaining multi-level planar maps in intelligent systems

John Albert Horst

Intelligent Systems Division, Manufacturing Engineering Laboratory
National Institute of Standards and Technology (NIST)
Gaithersburg, Maryland 20899
USA
horst@cme.nist.gov

Abstract

We present algorithms that allow an intelligent system to dynamically convert between two representations of spatial occupancy, namely, certainty grids and object boundary curves. These algorithms can be used to accomplish many real-time tasks of a mobile robot such as mapping, navigation, object recognition, and robot localization. For conversion from certainty grid to object boundaries, an edge linking algorithm [8] is appropriately modified. Certainty grid 'images' are transformed into a set of object boundary curves. The latter are expressed as oriented piecewise linear segments. Image processing techniques, such as edge detection, thinning, curve tracing, and linear approximation are employed with various modifications. Modifications include a new method for linear curve approximation that is simple, accurate, and efficient. This method monitors chord and arc length and its excellent performance is demonstrated against similar algorithms. The certainty grid to object boundary algorithm is tested against simulated noisy certainty grid maps. An algorithm to do the inverse operation, namely, to convert object boundary curves to an occupancy grid, is also presented.

1 Introduction

An intelligent system contains world model representations of entities necessary for the accomplishment of its goals. For example, a mobile robot must represent (implicitly or explicitly) the obstacles in its environment. Researchers have often chosen a single type of representation for all entities. We argue the utility of multiple representations of the same entity. The main objection to having multiple representations of the same entity in a real-time system is the problem of maintaining consistency between the representations. This problem is avoided when real-time conversion algorithms are defined and used. We present two such algorithms in this paper.

With processor and memory costs continuing to fall and the availability of parallel bus architectures, maintaining multiple representation types in real-time intelligent systems is becoming more feasible. The system benefits by being able to choose the representation most appropriate for the accomplishment of each task.

Two useful and complementary map representations are certainty grids [7] and object boundary curves. They are particularly useful for mobile robot control. For example, vectors normal to an object boundary would be difficult to get from a certainty grid, but relatively easy to obtain from an object boundary curve. Spatial occupancy information is gotten easily from certainty grids but not as easily from object boundary curves. Additionally, representing spatial occupancy in the form of object boundary curves is important for the detection of higher level features such as corners, curves, and lines. As a result, high level geometric features can be more easily computed, perceived, and updated and, for example, can be used by the mobile robot to reorient itself or to recognize objects. It is relatively easy to build and maintain a certainty grid which makes it a good local map. However, a certainty grid representation for a global map may require a forbidding amount of storage space, whereas, an object boundary curve representation is more compact without sacrificing accuracy or utility.

Since we seek to maintain these two particular representations of a dynamic occupancy map in *real-time* intelligent systems, we need *real-time* map conversion algorithms. We have developed two algorithms to do this real-time conversion, namely, certainty grid to object boundary (CGOB) and object boundary to occupancy grid (OBOG). Our claim of real-time performance is two-fold, 1) both algorithms are $O(n)$ where n is the number of pixels in the object boundary curves and 2) if the image processing segments of the algorithms are done on each pixel in parallel, the CGOB can execute in roughly five seconds or less for a standard sized image (i.e., 256^2 pixels) on standard computing hardware. The OBOG algorithm executes much faster than CGOB. These numbers have been determined experimentally. Significant speed ups can be made on this prototype code.

The CGOB algorithm we have developed has been tested against simulated certainty grids of various types corrupted by blurring and speckle noise (figure 6). The

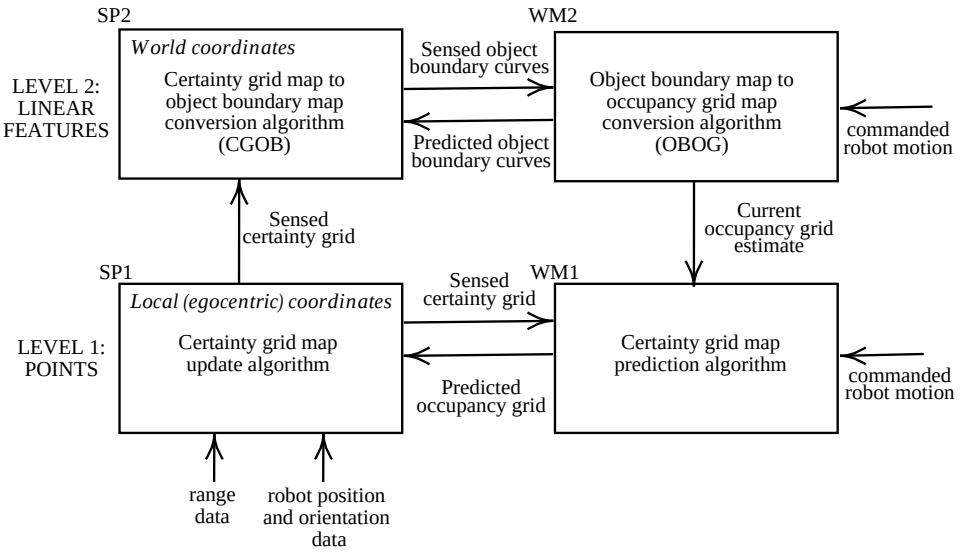

Figure 1: Sensory processing and world modeling components of an intelligent system showing functionality of map conversion algorithms.

object boundary curve points encode (in the ordering of those points) which side of the curve is occupied. The CGOB algorithm concludes with a piecewise linear curve approximation algorithm. The popular split and merge approach [6, 5] is known to be inefficient and several attempts to improve its efficiency come with an increase in complexity [8, 11]. We have developed a new approach that is accurate, simple, and efficient. We compare this new approach to some others in the literature.

The OBOG algorithm has been tested against object boundary curves of various types (figures 9 and 10). This algorithm is significantly simpler than the CGOB algorithm and can also be computed in parallel on a pixel processor.

2 Multi-level representations in an intelligent system

Hierarchical intelligent control, as described in [1], specifies a real-time, multi-level interaction of prediction and error formation. The certainty grid representation allows prediction of points (low level), whereas the object boundary representation allows prediction of lines and shapes (higher level). The CGOB and OBOG algorithms will allow both representations to simultaneously exist and be updated in a real-time hierarchical intelligent control system in a manner illustrated in figure 1. For example, robot range and position data can be used to update a certainty grid (a 'point' type of representation). Using CGOB, we convert this map to a set of object boundaries which can be considered to be a 'higher' level representation since we have now aggregated point features into linear features. These object boundary curves are then stored in the world model and can be used, for example, to do object recognition. Object recognition can be used to generate a more precise boundary map and OBOG can be used to create occupancy grid estimates. Knowledge of commanded motions can also contribute to better estimates.

3 The certainty grid to object boundary algorithm

We now describe the CGOB algorithm:

1) Create a raw edge grid using two orthogonal 5x5 gradient operators on a noisy certainty grid.
2) Threshold and thin the raw edge grid and use this result to compute arrays of 'predecessors' and 'successors' .
3) Group contiguous cells in the thinned edge grid, constituting contiguous edge cells.
4) Do local Gaussian smoothing on each group of points (to filter quantization noise).
5) Approximate the smoothed boundary points with contiguous line segments by monitoring change in chord length and path length.

3.1 Edge detection

We used two 5x5 orthogonal stochastic gradient operators [6] for computing the gradient. 3x3 operators didn't produce smooth thinned edges, so 5x5 operators were required. Stochastic gradient operators have the advantage of performance tailored to the expected signal to noise characteristics of the raw certainty grid. This signal to noise ratio (SNR) needs to be computed from a representative noisy certainty grid in order to be accurate. We performed our simulations with a somewhat low SNR of one.

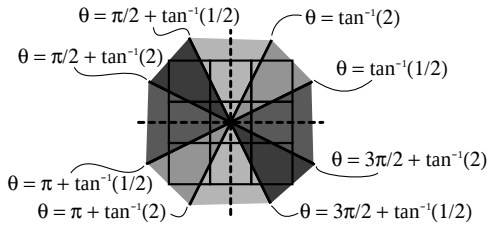

Figure 2: Regions of edge orientation in the neighborhood of a cell.

3.2 Thresholding and thinning

After edge detection, a thresholding operation is used to eliminate spurious edges. Our choice of the stochastic gradient operator made the choice of this threshold value less critical, since the stochastic gradient eliminates much of the noise.

The thinning step seeks to find the cells associated with the true edge and eliminate all others. The orientation and magnitude of each cell is examined. We classify the edge orientation as lying within one of four like-shaded regions in figure 2. The non-maximum suppression algorithm is employed for thinning and requires the following for a cell to be an edge:

1) the cell magnitude is a local maxima with respect to its two neighbors orthogonal to the edge orientation (see figure 2).
2) the difference in edge orientation of this cell with its two neighbors is less than threshold ($\pi/3$ worked well with our simulations)

If the cell passes both the threshold and the thinning test, it is stored in the thinned edge grid and the two neighbors are excluded from consideration as potential edge cells.

3.3 Curve linking

Now that the thinned edge grid is formed, we must order these cells into sets of ordered lists which describe the curves (closed, open, and forks) that we expect to see. We followed Nevatia & Babu by forming successor and predecessor grids that contain the chosen successor and predecessor for each cell. These predecessors and successors are then used to get linked lists of curves representing the object boundaries.

3.3.1 Predecessors and successors

The predecessors and successors for each cell in the thinned edge grid are chosen as follows:

1) Determine within which one of eight regions (illustrated in figure 2) the edge orientation lies. Edge direction is defined $\pi/2$ counterclockwise from the edge gradient direction. This means that occupied space is always to the right if moving along the edge in the edge direction.
2) With this information, define potential successors and predecessors (three each maximum) based on whether those potential cells are in the thinned edge grid.
3) If there are predecessors, record that fact. Any more information on predecessors is not necessary.
4) If there is only one successor (defined in the edge direction), just choose it.
5) If there are exactly two successors, and
   a) if the potential successor cells are not 4-neighbors, choose one with the greatest magnitude as the successor and store other as a fork.
   b) if the potential successor cells are not 4-neighbors, choose one as a potential fork only if its edge orientation differs by more than threshold (we used $\pi/3$). Otherwise, choose the nearest of the two as successor (Euclidean distance).
6) If there are exactly three successors,
   a) the one with maximum difference in edge orientation is chosen as a potential fork.
   b) successor is the closest one by Euclidean distance.

Optimum paths are not sought after as in dynamic programming techniques [2, 6], because of the increase of computation required.

The edge direction is defined as $\pi/2$ counterclockwise from the edge gradient direction and the edge direction defines the successor direction. Therefore, occupied space is on the right if one follows successors.

Using a 5x5 edge approximator, features of size five or smaller (depending on noise levels) will be missed or misinterpreted. This is probably why we were never able to create forks in our simulations.

3.3.2 Curve generation

Now that we have grids of successors, predecessors, and forks, we need to exploit these grids to obtain the ordered lists that constitute the object boundary curves in the certainty grid. Curve generation requires two serial passes through the grid.

1) Look for cells with a successor and no predecessor and store them as starters of open curves. At the same time, look for fork cells.
2) Start tracing at each starter cell (excluding fork cells). Eliminate the starter cell from further consideration. If the current cell has a successor and this potential successor cell has not been eliminated, store the successor into the ordered list. Delete each traced edge cell in the thinned edge map.
3) Start tracing at each fork only if the successor of the fork point successor has not been eliminated. Eliminate cells in fork curves.
4) Trace closed curves starting at the first cell encountered that is 'alive'. Eliminate cells from thinned edge grid as they are traced. Continue cycling through the entire thinned edge map.

3.4 Piecewise linear curve approximation

The map is now represented as sets of contiguous grid cells. Each set defines an object boundary curve. This is already a significant reduction in data storage from the certainty grid. However, further reductions can usually be made by approximating these curves with even fewer points. These points then represent the approximation to the object boundary.

One can, of course, use more sophisticated methods of fitting a curve to the set of obstacle boundary points using splines and higher order polynomials, but line segments have the advantage of simplicity. We avoided optimal linear approximations [9] since they a computationally much more expensive [10]. The trade-off of optimality for simplicity and speed we felt was reasonable.

Each object boundary is a list of contiguous cells constituting a curve in two dimensions. We wish to obtain a set of points that approximate that curve according to some criterion of fitness. One algorithm that does this is called 'split and merge' [3, 5, 6]. It uses a minimum mean squared error criterion. However, the split and merge method has several weaknesses,

1) It is inefficient since it may require forming approximation errors from the same points on the curve up to $(n-2)(n-1)/2$ times for an n point curve [8].

2) Errors can occur in the choice of the initial break point [11].
3) Attempts to eliminate these problems come with a substantial increase in complexity [11, 8].

The strengths of the split and merge algorithm are:

1) Simplicity.
2) It is controlled by a single, physically meaningful parameter (maximum deviation).
3) It generates a sparse set of points that well approximate the original curve (see figure 3).
4) The approximations are stable (according to the criteria defined in [4]).
5) It preserves symmetry.

We sought an algorithm that is efficient and minimizes mean squared error, but which preserves the strengths of the split and merge approach. Such requirements are largely satisfied by monitoring the relationship of chord length and arc length along the curve.

Here is the chord and arc length method for piecewise linear curve approximation:

1) Determine whether the curve is open or closed.
2) Do local Gaussian smoothing on the raw data to reduce quantization error.
3) Starting anywhere on the closed curve (at the first point on the open curve), compute chord length, C, and arc length, S, for each successive point and if $1/2\sqrt{S^2 - C^2}$ is greater than the maximum deviation parameter, declare the previous point to be dominant.
4) Merge points by testing if approximating points can be eliminated without exceeding the threshold on deviation.
5) Compute a (parameterized) least squares line to the points on the curve between and including the last two dominant points.
6) Find the point on the previous and current least squares fit lines that are closest to the previous dominant point.
7) Choose the midpoint between these two closest points as the latest approximating point.

Gaussian smoothing is suggested because the thinning algorithm often produces thinned edge pixels having one four-neighbor and one eight-neighbor (see figure 6).

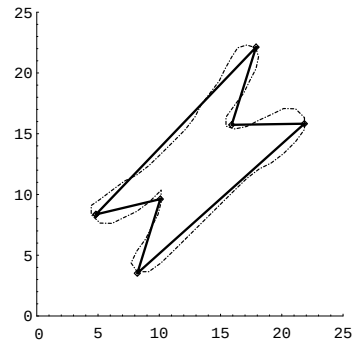

Figure 3: A Gaussian smoothed (window size = 5) digital closed curve with fitting by chord and arc length with least squares

We illustrate in figure 4 the maximum error of the various algorithms for the digital closed 'chromosome' curve used in figure 3. Note that as the number of approximating points increases, the maximum error for the split and merge and the chord and arc length methods converge. Figure 5 illustrates the superior efficiency of the chord and arc length approach over the split and merge algorithm and its insensitivity to the number of approximating points (i.e., 'tightness' of fit).

4 The object boundary to occupancy grid algorithm

The OBOG algorithm we now describe is significantly simpler to state and compute than the CGOB algorithm. This is not surprising, since object boundaries are a more compact representation than certainty grids and can be considered to be at a higher level in the hierarchy of the intelligent system. This is because object boundary curves are beginning to organize space into connected lines whereas certainty grids organize space into disconnected points. More work is required to discern this connectivity of points into lines and then to make appropriate approximation of those lines. Input to the OBOG algorithm are the object boundary curves which define spatial occupancy. Each of these curves is oriented so that as one moves along the curve, occupied space is to the right. For each cell in the grid:

1) Find the point on the set of boundary curves that is closest to the cell. This will either be a vertex on the curve or a point within a single line segment joining vertices. If there are two or more points on the curves that are at the same distance from the cell, one can be picked randomly without error.
2) If the point on the curves closest to the cell is a point within a line segment, compute the cross product of the vector from the cell to the point and the vector of the (oriented) line segment. If the sign of the cross product is positive, the cell is in unoccupied space; if negative, the cell is in occupied space.
3) If the point (on the curves) closest to the cell is an endpoint of two contiguous line segments, form the dot product of the vectors formed by the two contiguous (oriented) line segments.

3a) If the sign of the dot product is positive, compute the cross product of the vector from the cell to the endpoint and the vector of either of the (oriented) line segments. If the sign of the cross product is positive, the cell is in unoccupied space; if negative, the cell is in occupied space.

3b) If the dot product is less than or equal to zero, form two cross products of the following pairs of vectors a) the vector from the cell to the endpoint and the vector of the first of the two (oriented) line segments and b) the vector from the cell to the endpoint and the vector of the second of the two (oriented) line segments.

3c) If these cross products both have positive sign, the cell is in unoccupied space. If both have negative sign, the cell is in occupied space.

3d) If these cross products have different signs, form the vector of the sum of two unit vectors in the direction of the vectors formed by the two line segments on the curve. Compute the cross product of the vector from the cell to the endpoint and this sum vector. The cell is in unoccupied space if this sign is positive and is in occupied space otherwise.

5 Simulation and application

For testing the CGOB algorithm, we used simulated grid maps with added Gaussian blurring and speckle noise. Figure 6 shows an example of such a noisy grid. Note that the final approximating line segments are superimposed. Here we see that both open or closed curves can be detected and that data reduction from the certainty grid to the obstacle boundary points is by a factor of about 145 in this example. Figures 7 and 8 show the raw edge grid and the

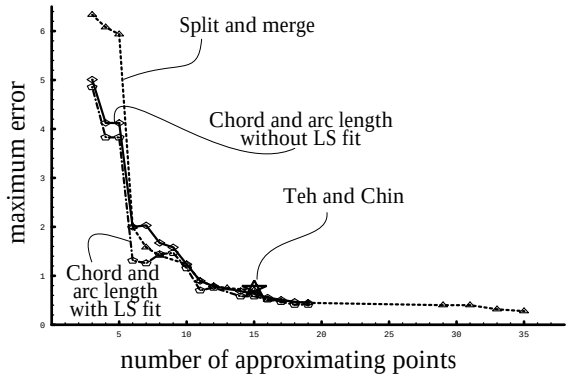

Figure 4: Maximum errors as a function of the number of approximating points for several algorithms using the digital (unsmoothed) curve of figure 3.

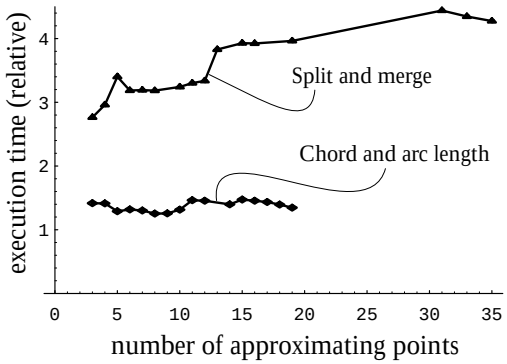

Figure 5: Relative execution time as a function of the number of approximating points for the chord and arc length algorithm and the split and merge method using the curve of figure 3. Note that in addition to executing faster, the chord and arc length approach has relatively flat execution times with respect to the number of approximating points compared to the split and merge approach.

thinned edge grid. Programming and testing was done in *Mathematica*™ and 'C'.

For testing the OBOG algorithm, we used object boundary maps like those in figures 9 and 10. Note that the initial line segments constituting the object boundaries are superimposed over the resultant occupancy grid. Programming and testing was done in *Mathematica*™. Conversion to the 'C' language and implementation on a mobile robot is in progress.

The initial application for these algorithms was to maintain maps of an underground coal mine for a computer-controlled mining machine [13]. The certainty grid is particularly suitable for specifying the presence or absence of coal. Obstacle boundary curves are appropriate as a global map because it is of equal resolution as the certainty grid but requires much less storage space.

6 Conclusion

Some of the uses and advantages of having the CGOB and OBOG algorithms in a real-time robotic system are:

1) The simultaneous use and dynamic availability of two forms of spatial occupancy representation, namely, certainty grids and object boundary curves.
2) Certainty grids provide a fast response to some queries that would take longer to get with object boundary curves (e.g., Is there an object at (x,y)?). The latter would have faster response to other types of queries (What is the normal vector to the boundary at (x,y)?).
3) Certainty grids require much more storage space than object boundary curves. This fact might suggest the former for local, egocentric maps and the latter for global maps. Using the OBOG algorithm, the system can bring in sections of the global map (as obstacle boundary curves) and convert them into certainty grids, as needed.
4) Object boundary curves can be considered a 'higher level' representation in an intelligent hierarchical system, since, for example, it is easy to detect geometric shapes using object boundary curves.

The most computationally expensive portions of the CGOB algorithm are edge detection, thresholding, thinning, and the generation of predecessors and successors. These portions are designed to be computed on image processing hardware. Similarly, the entire OBOG algorithm can be computed using image processing hardware. As we stated in the introduction, the execution time of the CGOB algorithm should be fast enough for a variety of real-time mobility tasks.

More work is required in at least two areas. 1) We need to execute the algorithms on image processing hardware. 2) We suggest that the certainty grid be maintained as a local map and a set of obstacle boundary curves as a global map. This requires that there be a way to integrate a local obstacle boundary map into the global obstacle boundary map.

References

[1] Albus, J. S., "A Theory of Intelligent Machine Systems", Proc. of IEEE Intelligent Robots & Systems 1991 -- Intelligence for Mechanical Systems, November 1991.

[2] Bellman , R. E. and Dreyfus, S., *Applied Dynamic Programming*, Princeton, N.J..: Princeton University Press, 1962.

[3] Duda, R. O. and Hart, P. E., *Pattern Recognition and Scene Analysis*, New York: John Wiley, 1973.

[4] Fischler, Martin A. and Bolles, Robert C., "Perceptual organization and curve partitioning", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8 No. 1, January 1986, 100-105.

[5] Grimson, W. E. F., *Object Recognition by Computer: The Role of Geometric Constraints*, Cambridge, Massachusetts: The MIT Press, 1990.

[6] Jain , A. K., *Fundamentals of Digital Image Processing*, Englewood Cliffs, N.J.: Prentice Hall, 1989.

[7] Moravec, Hans P., "Sensor fusion in certainty grids for mobile robots", AI Magazine, Summer 1988, 61-74.

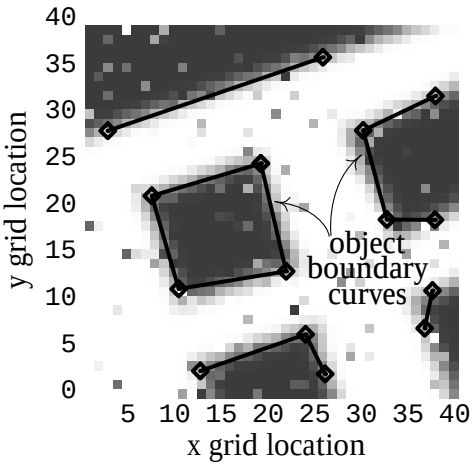

Figure 6: A noisy certainty grid with final approximating points superimposed.

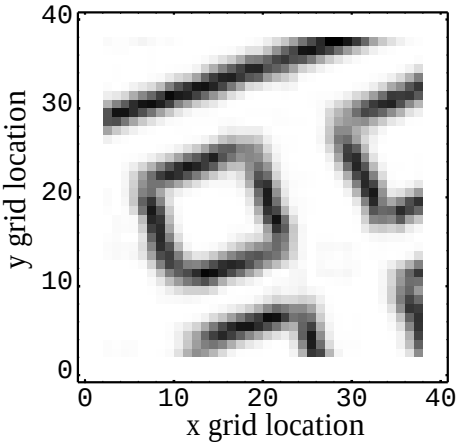

Figure 7: The raw edge grid for noisy certainty grid of figure 6.

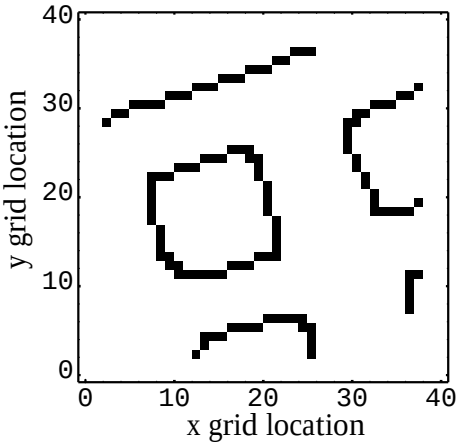

Figure 8: The thinned edge grid for noisy edge grid of figure 7.

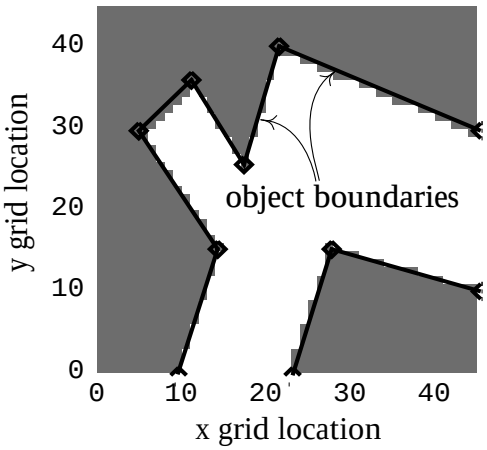

Figure 9: An example set of object boundary curves with the resultant occupancy grid overlaid

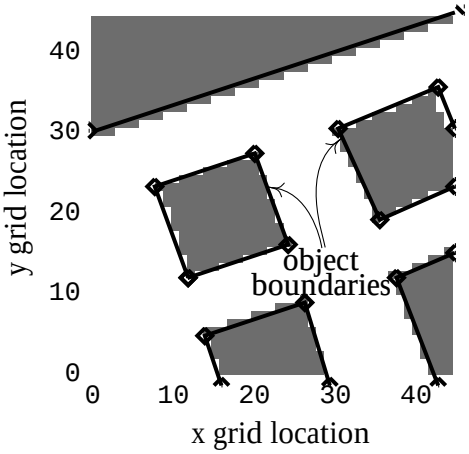

Figure 10: Another example set of object boundary curves with the resultant occupancy grid overlaid

[8] Nevatia, R. and Babu, K. R., "Linear Feature Extraction and Description," *Computer Graphics and Image Processing* **13**, 257-269.

[9] Pavlidis, Theodosios, "Polygonal approximations by Newton's method", *IEEE Transactions on Computers* **C-26**, No. 8, August 1977.

[10] Ramer, Urs, "An iterative procedure for the polygonal approximation of plane curves", *Computer Graphics and Image Processing* **1**, 1972, 244-256.

[11] So, W. C. and Lee, C. K., "Invariant line segmentation for object recognition", *Proceedings of IECON '93*, Maui, Hawaii, Nov. 15-19, 1993, 1352-1356.

[12] Teh, Cho-Huak and Chin, Roland T., "On the detection of dominant points on digital curves", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. II No. 8, 1989, 859-872.

[13] Horst, J. A. and Tsai, Tsung-Ming, "Building and maintaining computer representations of two-dimensional mine maps," Proceedings of the 3rd International Symposium on Mine Mechanization and Automation, Golden, Colorado, June 1995, pg. 22-1.