Representational World Modeling

A statistical cell-based map representation that supports unrestricted path planning and collision avoidance for indoor operations.

by H.R. Everett

The preceding article of this two part series (*TRP*, Winter 1996, p. 15) examined a very basic reactive control scheme that could be used to intelligently direct the motions of the prototype security robot ROBART I operating in very structured and relatively obstacle-free surroundings. The reactive control approach couples real-time sensor information to motor response without the use of intervening symbolic representations that model the environment. The robot could move as required to any given room simply by maintaining a relative awareness of the room inter-relationships along the hallway, but had no absolute sense of its own location in the X-Y floorplan. In addition, reflexive avoidance of potential obstacles could easily interfere with effective execution of the navigation algorithm, sometimes with non-recoverable results.

In this issue we review a simplistic representational world modeling approach originally employed on ROBART II. This much improved second-generation robot was equipped with 132 external sensors, to include a 360-degree *navigational sonar array* for initial map generation and position location, as well as a *collision avoidance sonar array* (Figure 1). A distributed architecture of 13 on-board microprocessors handled dedicated tasks such as drive motor, sonar, head position, and speech control, linked via a serial RF data link to a desk-top 386-based PC that performed all high-level tasks associated with world modeling.

Providing such a model-based navigational capability involves the implementation of an appropriate map representation, the acquisition of information regarding ranges and bearings to nearby objects, and the subsequent interpretation of that data in building and maintaining the world model. Each of these issues will be treated in some detail in the following sections.

Selecting a Map Representation

The simplest map representation is a two-dimensional array of cells, where each cell in the array corresponds to a square of fixed size in the region being mapped. Free space is indicated with a cell value of zero; a non-zero cell value indicates an object. The most compact form of a cell map consists of one bit per cell and thus indicates only the presence or absence of an object, such as the example presented by Balch (1996) in the last issue of *The Robotics Practitioner.*

By using multiple bits per cell, additional descriptive information can be represented in the map, to

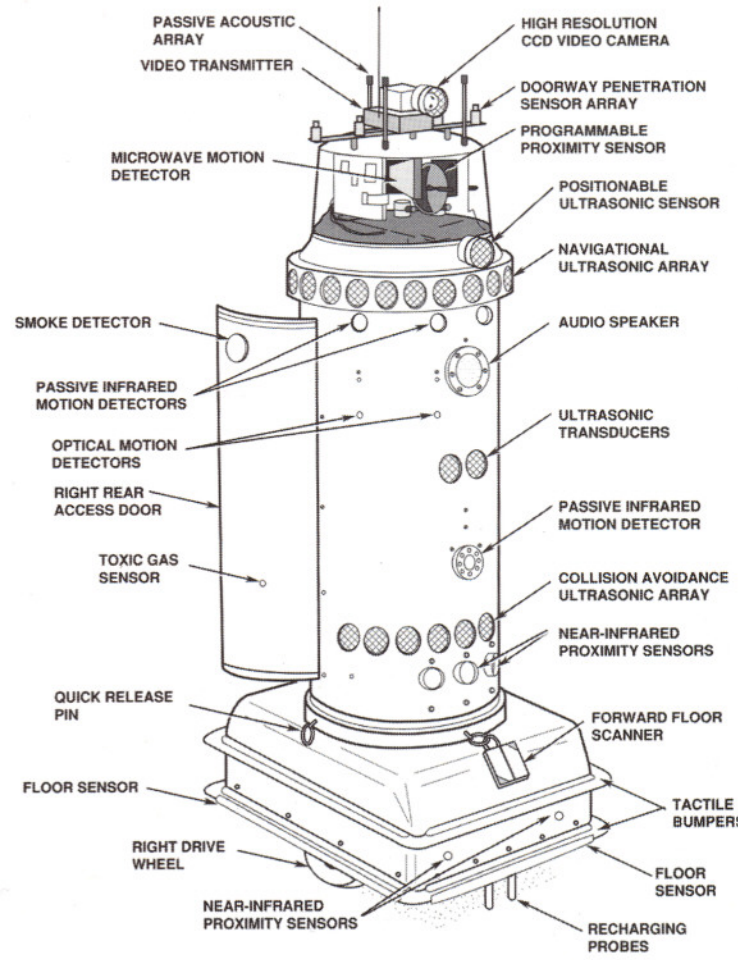

Figure 1: *Location of the various security, navigation, and collision avoidance sensors employed on ROBART II (1982-1992).*

include the probability of a given square being occupied. This feature is useful when the precise locations of objects (and even the robot itself) are unknown. Memory usage is independent of map content, so cluttered surroundings are not a problem. The resolution of the map is only as good as the square size, however, and doubling the resolution quadruples the memory requirements.

A slightly more sophisticated and elegant approach is to use a quadtree representation (Fryxell, 1987). The world map begins as a square which is in turn subdivided into four smaller squares. Each of these squares is in turn recursively subdivided (down to the map resolution if necessary) until the region occupied by the square is homogeneous (all object or all free space). For an uncluttered environment, a substantial savings in memory usage is achieved with a decrease in find-path execution time, since the effective map size is smaller. In highly inhomogeneous environments, however, memory usage can increase beyond that of the simple cell map, thus negating the primary advantage of the quadtree.

A third technique uses polyhedra and curved surfaces or geometric primitives to represent objects in the workspace (Lozano-Perez, Wesley, 1979; Brooks, Lozano-Perez, 1983). Such maps are quite compact, and with no inherent grid, the locations of objects can be more precisely entered into the model. These maps are also easily extended into three dimensions, in contrast to a cell-based map where memory cost would be prohibitive. Updating the map with real-world data is difficult, however, as it is hard to accurately glean polygonal information from inexpensive sensors mounted on a mobile robot. Statistical uncertainty of the existence of objects is difficult to implement as well.

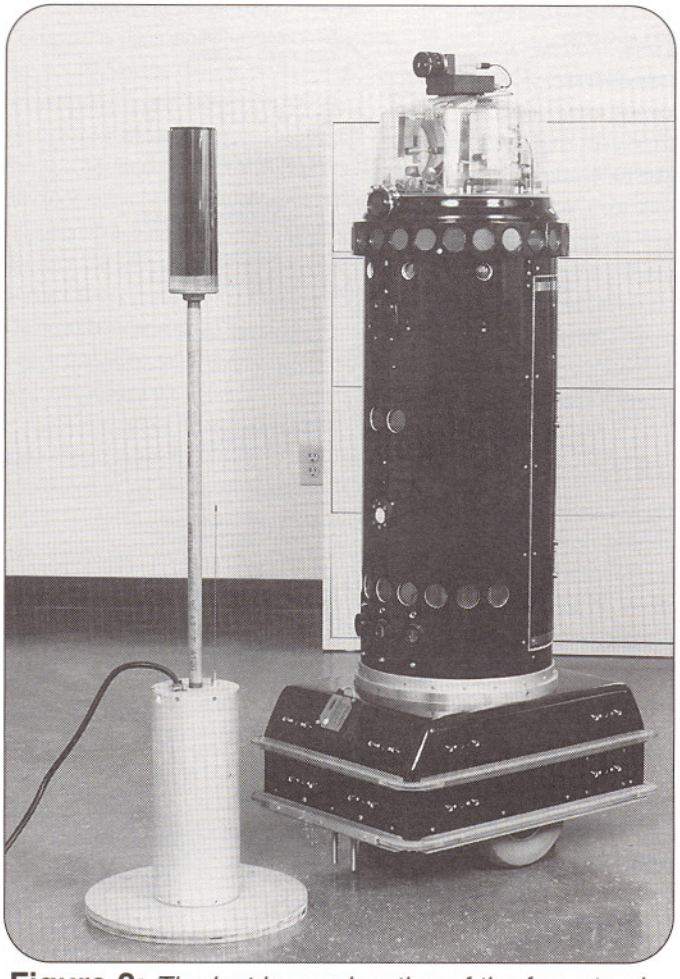

Figure 2: *The last known location of the free-standing recharging station is represented in the world model as a special case of a transient object.*

Brooks (1983) has devised a scheme based on generalized cones. Rather than map both obstacles and free-space, only the areas the robot can freely traverse are mapped; the robot is required to stay within these *freeways*. Paths can be quickly found using this method and do not "hug" obstacles as is typically the case with many other algorithms, thus decreasing the likelihood of collisions due to cumulative dead reckoning errors. This algorithm does not work well in cluttered environments, however, and decomposing free space into generalized cones can be computationally expensive.

Following an examination of these and various other alternatives, a cell-based map was adopted for use on ROBART II for a number of reasons (Everett, et al., 1990):

- The area described by the map is a bounded space (i.e., a building interior), where a relatively coarse grid (3-inch resolution) can be used.
- Objects of unknown configuration are easily added. This feature is of particular importance since low-resolution ultrasonic sensors are used for map updating.
- The traversibility of a square can be statistically represented and easily changed.
- Unique coding of predefined entities (i.e., doorways, recharging station (Figure 2), recalibration sites) is easily supported.
- A simple Lee maze router (Lee, 1961) can be used for path planning.
- The map can be accessed and updated quickly.

Acquiring and Incorporating Range Data

Real-time range data acquired by both the *navigational* and *collision avoidance sonar arrays* (see again Figure 1) must be appropriately entered into the world model as the robot is moving. This seemingly trivial operation turns out to be more difficult than expected due to problems associated with the operation of ultrasonic ranging systems in air, the subject of an earlier article in the Fall issue of *The Robotics Practitioner* (Everett, 1995b). Typical problems include temperature dependence, which has an impact on range accuracy, and beam dispersion, which contributes to angular uncertainty. Specular reflection from target surfaces can cause additional errors, and adjacent sensor interaction requires the transducers in the array be individually fired rather than simultaneously. Finally, the slow speed of sound in air results in marginal update rates, as well as the need for successive coordinate transformations to account for displacement due to robot motion during the sequential firing of all transducers in the array (Everett, 1985).

Consequently, the effective interpretation and use of inherently questionable sonar data is critical to achieve a reasonably accurate representation of surrounding obstacles. Moravec and Elfes (1985) of CMU describe a scheme for mapping sonar range returns using probability distribution

functions. For each sensor reading, the assigned probability of an object being at the exact indicated range and bearing is considered high, and decreases radially around that point according to a pre-specified distribution function. In addition, a second distribution function characterizes the "emptiness" of cells between the sensor and the returned range. Points near the sensor have a high probability of being unoccupied, with decreasing probability for those points closer to the indicated range or off the beam axis. The technique was applied to a map where the state of occupancy for all cells is initially marked as unknown. The CMU robot was moved to various vantage points in the room, with several sonar readings taken at each point and averaged to create the probability map.

Fryxell (1987) as well as Beckerman and Oblow (1988) also used probability schemes for mapping sonar data. Fryxell took sonar readings (modeled as rays) from different places in the environment and then constructed two arrays, one observing the number of times each cell was "hit," and the other observing each time a cell was "missed." A voting procedure combined both maps to create the final map, with each cell marked as either occupied or unoccupied.

Beckerman and Oblow (1988) used a similar method but modeled the sonar beam as a cone subtending an angle of 18 degrees. The reduced effective beamwidth (18 versus 30 degrees) is achieved by employing a phased array consisting of four transducers (Everett, 1995a), the array being sequentially repositioned mechanically to achieve the desired coverage. As with Fryxell (1987), the robot was moved to various vantage points in the room to make static sonar observations. These data were saved in auxiliary buffers and used to update a cumulative map, with each cell labeled as *conflicting, unknown, occupied,* or *empty*. (A *conflicting cell* occurs when one or more sonar readings intersect such that one marks the cell as occupied while the other marks it as empty.) After all nonconflicting data had been integrated into the cumulative map, conflicting cell status was resolved through pattern analysis of the original data. This technique generated maps similar to those created by Fryxell's method, but with better resolution even though fewer sonar readings were taken.

A faster and less computationally expensive variation of these statistical representation schemes was implemented on ROBART II. By using a simplified probability distribution and range-gating fixed arrays of sonar sensors, the mapping process can take place in real time while the robot is in motion. Two different mapping procedures are used, one for creating the initial map and another for updating the map during the execution of a path. In addition, two distinct classes of objects are defined: 1) *permanent objects*, which are essentially fixed in place, and, 2) *transient objects*, which tend to move around.

The world model contains positional information about all the known objects in the environment, and may be either robot or human generated, or a combination of both. In either case, only relatively immobile (hence the term *permanent*) objects (i.e., walls, desks, filing cabinets) are recorded during the initial map generation procedure. An observed correlation between the height of an object and its degree of long-term positional stability heavily influenced the vertical placement of the *navigational* and *collision avoidance sonar arrays* during construction. In other words, the taller an object, the more likely it was to remain stationary, hence the navigational array was placed as high as possible (see again Figure 1).

Objects likely to be transitory in nature (i.e., chairs, trash cans, carts) are not recorded in the original map and present a problem during actual path execution, giving rise to the need for an effective collision avoidance capability. For reasons cited above, the collision avoidance sensors function best when situated nearer to the floor surface. Special cases of *transient objects* include doorways, which can be open or closed, and the robot's battery recharging station shown earlier in Figure 2.

Initial Map Generation. To generate the initial map, the robot moves very slowly around the room boundaries while firing all 24 transducers in the upper navigational array. The sonar beams are modeled as rays and range-gated to six feet. If the indicated distance is less than six feet, the probability value assigned to the cell corresponding to that location in the map is incremented once. After the room had been completely traversed, the cell values are thresholded using a special Map Editor utility to remove noise and achieve the best map definition. The resulting map contains the known permanent objects in the room as seen by the robot and can next be manually edited to add additional features, such as hidden lines, doorways, etc. Each object in the map is then automatically "grown" by half the width of the robot in order to model the robot as a dimensionless point during subsequent find-path operations (Lozano-Perez, Wesley, 1979).

Free space is represented by an array value of zero (Table 1) and is shown in white. *Permanent* objects are displayed as light gray, whereas *transient* objects are displayed as black. The dark gray area surrounding each *permanent* object is the "growth" which allows the robot to be modeled as a point. *Permanent objects* and their associated growth are thus created under human supervision and cannot be later erased by the robot during path execution.

Modeled Entity	Cell Value
Free Space	0
Transient Object	1-16
Current Location(START)	238
Destination Goal (DEST)	239
Recharging Station	250
Doorway (open)	252
Permanent Object Growth	253
Permanent Object	255

Table 1: *Assigned cell value for various entities represented in the world model. Range of values (1-16) for transient objects reflects the probability of cell occupancy.*

Constructing a map from the ground up (no pun!) using the Map Editor can be tedious and time consuming. Often, however, a CAD drawing of the building has already been created. A translator was therefore written to convert an AutoCAD™ *Drawing Interchange File* into a bit-mapped model that the path planner can use. The only user input required is the desired cell size (resolution). Special-case items such as doors, recharging stations, and recalibration sites (see again Table 1) are manually entered after the conversion using the Map Editor as before, but any permanent entities such as structural walls and furniture are dealt with automatically.

Dynamic Map Maintenance. A different mapping approach is used when entering data from the collision avoidance sonars into the world model during actual execution of a path segment: only the center five transducers in the array are activated. If a given sensor return shows an object within five feet, the certainty value assigned to the cell at that indicated location is incremented twice as shown in Figure 4, up to a specified maximum (typically 16). In addition, the probability values assigned to each of the eight neighboring cells are incremented once to take into account uncertainties arising from the dispersion angle of the ultrasonic beam. Cells previously marked as being permanent objects or growth, however, are left untouched since they will always be avoided by the path planning search algorithm.

In addition, each time a sonar range return is processed, all the cells within a cone ten degrees wide and four feet long (or less if an object appears within four feet) have their assigned values decremented by one. This action, first introduced by Moravec and Elfes (1985), effectively erodes objects no longer present and also serves to refine the representation of objects as the robot moves through a series of new perspectives. Transient objects are erased from the map at a slower rate than they are entered, so the system tends to err on the side of not running into obstructions. As with object addition, permanent obstacles and growth are left untouched.

Figure 5 shows a three-dimensional bar chart depiction of such a map, where the height of each bar is proportional to the probability that the given cell is occupied. The probability cluster at point A marks the location of a chair which had been placed at the right-hand end of the room. The robot was told to execute a rectangular path, causing it to completely circumnavigate the chair. After the robot's first pass around the room, the chair was moved to point B and the path repeated, with the resulting probability distribution as shown in Figure 6. Note the robot quickly recognizes the new location of the chair, while the probability of an object being present at point A has been significantly decreased. In addition, two more objects can be seen at points C and D. Objects which do not change position over successive viewings are further reinforced until their associated probability functions saturate.

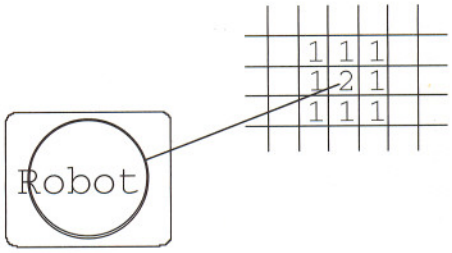

Figure 4: *For any objects detected within five feet, the certainty value assigned to the cell at the indicated sonar range and bearing is incremented twice, while the eight neighboring cells are each incremented once.*

Path Planning

There are a number of operations the path planning algorithm must address to get the robot from point A to point B:

- Find a clear path to the desired destination.
- If no path exists, then return a value of FALSE to the calling program.
- Retrace the search to create a list of straight-line segments describing the path.
- Create the necessary movement commands and execute the path.
- If the path is successfully executed then return successful status.
- Otherwise, plan a new path to the destination.

The original path planner was based on the Lee (1961) path connection algorithm with the cell coding enhancements suggested by Rubin (1974). The basic search algorithm begins by "expanding" the initial cell corresponding to the robot's current position in the floor map (i.e., each unoccupied neighbor cell is added to

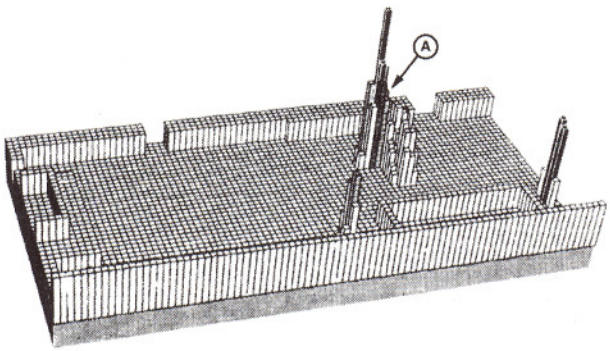

Figure 5: *Probability distribution showing the perceived location of transient objects in the environment. Note representation of chair at A.*

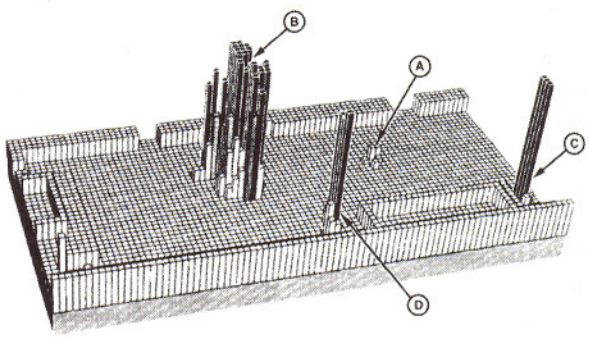

Figure 6: *Probability distribution after moving chair from location A to location B.*

the *expansion list*). Then each cell on the *expansion list* is expanded. This process continues until the destination cell is placed on the expansion list, or the list becomes empty (in which case no path exists). Details of these operations are discussed in the following subsections.

Finding a Path. As discussed, the world map contains a byte for each grid square in the room, where the size of a square can range from one inch up to several feet, depending on the room size and desired resolution. The path planner first makes a working copy of the map, in which the *Find-Path* routine stores two special bytes (see again Table 1), one indicating the current location of the robot (*START*) and the second indicating the desired destination (*DEST*). During the search process the algorithm looks for the floor cell containing the *DEST* byte, while during the backtrack process it looks for the *START* byte.

Information about the source cell (i.e., *START*) such as X-Y location and cost is then put onto a *frontier* list consisting of those points on the outer edge of the search envelope that are candidates for the *expansion* process, to be discussed below. Putting the source cell on the *frontier* list *seeds* the algorithm so it has a cell to expand. The algorithm then enters a loop which terminates only when there are no more cells on the frontier list, or when a path has been found. (If the frontier list is empty, then no path is possible and the search fails.)

The first step inside the loop is to find all the cells on the frontier list with minimum cost and put them on the expansion list. When a cell is placed on the expansion list, a value (arrow) indicating the direction to the parent cell is stored in the map array. Once the destination cell has been reached, retracing the path involves merely following the directional arrows back to the source. During this process only those points representing a change in direction (i.e., inflection points) are recorded, and the entire path is completely specified by the straight line segments connecting these inflection points. The resulting path is guaranteed to be the minimum distance path.

The minimal distance path, however, is not necessarily the "best" path. Sometimes it is more desirable to minimize the number of turns or to maximize the distance from obstacles, for example. The search strategy can be altered accordingly by assigning a cost to each cell prior to adding it to the expansion list; only the minimum cost cells are then expanded. This is known in the literature as an A* search (Winston, 1984). The *cost* of a cell is typically some computation of how *expensive* it is to look at a certain cell. In a typical A* search, the cost is set equal to the distance already traveled from the starting point to the current cell, plus the straight line distance from the current cell to the destination cell. This value is guaranteed to be less than or equal to the actual total distance from the source to the destination. This particular cost function tends to make the search expand in a direction towards the goal, which usually decreases the search time.

"The minimal distance path is not necessarily the best path."

The expansion process begins once the minimum-cost cells have been transferred from the frontier list to the expansion list. The expansion process looks at all the neighbors of each cell on the expansion list. Each unoccupied neighbor that is not on either the expansion or frontier list is placed on the frontier list. If the destination cell is reached (as will be discussed later), a solution path has been found and the algorithm terminates, otherwise termination occurs when all cells on the current expansion list have been expanded. A value of FALSE is returned in this latter case, indicating the destination was not reached during the current expansion and further searching of the new frontier list (as updated by the expansion process) will be necessary.

When selected, each cell on the expansion list is first checked for possible expansion, as the only valid candidates for expansion are those cells whose assigned byte in the floor map is zero. If the assigned certainty value is not zero, the cell may be occupied by a *transient* obstacle detected by the robot's sensors. In such a case the certainty value is simply decremented and the cell put back onto the frontier list for later expansion. This technique is used to bias the algorithm towards a clear path (if one exists) in preference to a cluttered path, as decrementing the non-zero certainty values slows down the expansion process in the presence of perceived obstacles. If no clear path is found, however, the robot may still be able to traverse the cluttered path. This ability to eventually "eat through" the representation of *transient objects* assists the path planner in finding a path even in the presence of faulty or inaccurate sensor data.

If the current floor map cell value is zero, indicating the cell can be expanded, then each of the cell's four orthogonal neighbors (i.e., north, east, south, and west) is examined to see if it is occupied or unoccupied. If a neighbor cell contains the special byte *DEST*, a path has been found, the X-Y location of the cell is saved, and *TRUE* status is returned. Otherwise, if the cell is unoccupied, it is placed on the frontier list. (If occupied it is simply ignored.) Finally, information used by the backtracking routine (basically an arrow indicating the direction to the neighbor's parent) is stored in the floor map cell corresponding to the current neighbor. Control is then returned to the top of the loop, and the expansion process is repeated with the updated frontier list as previously discussed.

Backtracking. Once the desired destination cell has been found, the *backtracking* process (also called

retracing or segmentation) creates the list of path segments that describe the path, based on the contents of the current floor map following the *Find_Path* operation. The procedure is very simple. Starting with the destination cell, the following steps are performed:

1. The direction arrow of the current cell is followed to the adjoining (parent) cell.
2. If the new cell contains START, then the algorithm is done.
3. If the direction arrow of the new cell is the same as the current arrow, the new cell becomes the current cell and the routine returns to step 1.
4. Otherwise the direction has changed, the current X-Y coordinate must be added to the path segment list, and the segment counter updated.

The output of the backtracking routine is thus a list of X-Y coordinates describing the waypoints through which the robot must pass in order to reach the destination.

Path Execution. Once a path segment list has been specified, the robot must follow the route described to reach the desired goal. For sake of simplicity, each segment of the path is executed individually by turning the robot to the required heading and then performing a straight-line move of the needed distance. Control is then passed to the segment execution routine. (In subsequent robots, consecutive straight-line path segments are joined by filet arcs to facilitate continuous motion.) This procedure returns a status condition indicating whether or not the robot was able to successfully execute the segment. If successful, then the next path segment (if any remain) is executed. During segment execution the path planner monitors status packets sent back by the robot, to include:

- A *move-complete report* indicating the robot has finished moving the desired distance.
- An *obstacle report* indicating the robot has stopped because an obstacle is in its way.
- A *dead-reckoning report* of current position.
- A *collision-avoidance sonar packet* updating the current map representation.

The software loops until one of the return conditions (i.e., *move-complete* or *obstacle*) is met.

Collision Avoidance

For a mobile robot to be truly autonomous, it must cope with the classic problem of avoiding an unexpected, unmapped obstacle. Initial approaches involved the development of a second localized *relative map* which represented the locations of objects detected in front of the robot while traversing a path segment (Harrington, Klarer, 1987; Crowley, 1985). When range to an obstacle fell below a critical threshold, robot motion was halted and an avoidance path around the obstacle was planned using the smaller relative map. In this approach, however, the relative map is very transitory in nature, created at the beginning of each move and discarded at the end. The only information in the map is obtained from on-board range sensors while the robot is in motion. Since there is no memory of previously encountered obstacles, no learning curve exists, and several avoidance maneuvers may be required to complete the path if the area is congested.

Another possibility would be to actually encode the position of newly detected transient objects into the original "absolute" map while a path is being executed. This scheme has the advantage that all prior information learned about the environment is also available. Thus, in addition to avoiding newly detected objects, the path planner also knows about previously modeled obstacles which may be out of sensor range or occluded in some way. However, unlike the first method, this technique tends to produce cluttered maps over a period of time, due to: 1) the transient nature of some objects, 2) erroneous sensor readings, and 3) uncertainty in actual robot position and heading. Eventually the workplace may appear impassable as far as the model is concerned.

A modified approach was adopted in the case of ROBART II to combine the best of both the relative and absolute schemes. Object space is subdivided into the two categories of *permanent* and *transient* objects as previously discussed. All collision avoidance sensor information is statistically represented in the absolute map, based on the number of times something was seen at a given cell location. Conversely, when a previ-

Figure 7: *Photo of the area diagramed in Figure 8 with the robot situated at Point A.*

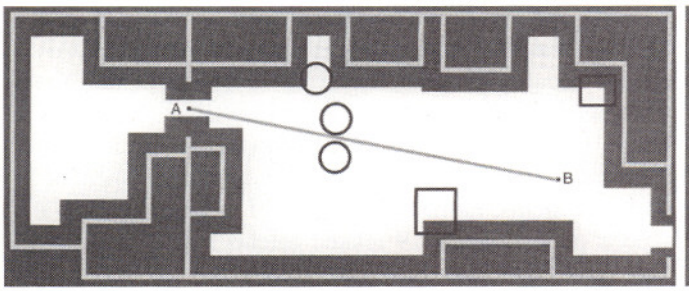

Figure 8: *Overlaid circles and rectangles show the actual locations of as yet undetected transient objects.*

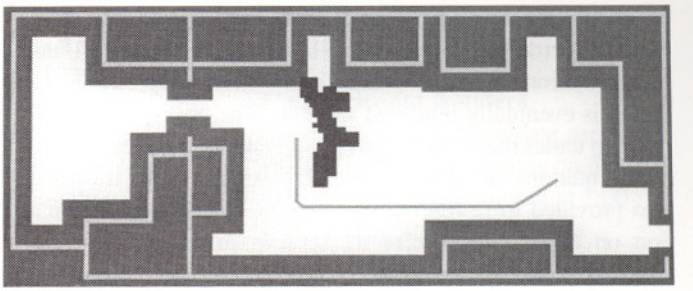

Figure 9: *Avoidance maneuver generated to clear the row of cylinders shown in Figure 7.*

ously modeled object is no longer detected at its original position, the probability of occupancy for the associated cell is decreased; if the probability is reduced to zero, the cell is again regarded as free space.

On completion of a path, all the *transient* cell probabilities are decreased by a small amount. This forced "amnesia" helps to eliminate sensor noise, and over a period of time causes all *transient* objects to be erased from areas seldom visited. This feature is advantageous in dynamic environments where the probability that the position of *transient* objects has changed is proportional to the amount of elapsed time since that object was last mapped by the robot.

The distinction between *permanent* and *transient* objects is an important feature that is largely responsible for the robust nature of the modeling scheme. *Permanent* objects remain in the model as a baseline from which to restart if the model for some reason becomes overly congested and must be flushed; only the *transient* objects are deleted. In addition, the path planner will always avoid permanent objects and their associated growth, whereas if necessary, the planner can penetrate the temporary growth surrounding transient objects in an attempt to find a path. This ability was found to be necessary, because in congested environments, the growth operation often closes off feasible paths due to inaccuracies inherent in the range data. The cost of traversing transient growth increases linearly in the direction of the perceived object location to minimize chances of a collision.

For purposes of illustration, we will review step by step the actions of the path planner and subsequent motions of the robot in executing a simple path through an obstructed area. A photograph (Figure 7) of the region reveals the presence of several unmapped obstacles; the positions and outlines of each have been graphically overlaid on top of the map in Figure 8 for sake of convenience. As the path planner initially has no knowledge of these objects, the resulting path is a straight line from point A to point B through the area occupied by the cylinders.

During the execution of this path segment, the *collision avoidance sonar array* detects the row of cylinders and begins altering the cell probabilities to reflect the perceived obstructions. When the robot has advanced to within the critical collision threshold (22 inches), forward motion is halted. At this point all the newly mapped objects are temporarily grown (for maneuvering clearance) in preparation for the path planning operation. This *transient growth* is removed after the new path has been found (Figure 9). The black areas represent the *transient obstacles* detected during the course of the first move. Upon executing the revised path, the robot discovers the cart and plans another avoidance maneuver (Figure 10). Upon reaching its destination B (Figure 11), the robot plans a path back to the original starting point A, this time avoiding the newly discovered obstacles.

Summary

This article has reviewed a very basic representational world modeling capability implemented on an autonomous mobile security robot. A number of simplifications were made to facilitate robust operation on a desktop 386-based PC:

- A distinction was made between *permanent* and *transient* objects.
- All objects were appropriately "grown" to allow the robot to be modeled as a dimensionless point.
- Objects were treated as their projection on the X-Y plane.
- An abbreviated statistical representation was employed to compensate for errors in both sensor data and robot position.
- Robot motion was restricted to point-to-point moves.

The principle difficulty associated with such a representational world modeling scheme is accumulated dead-reckoning

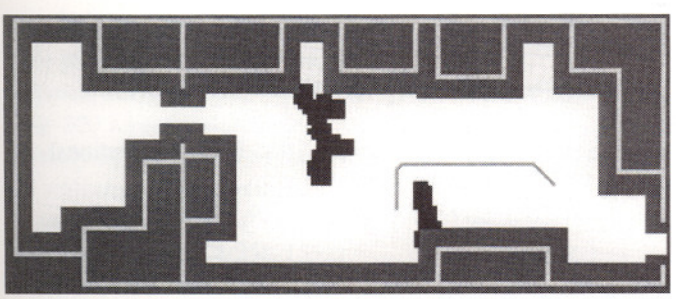

Figure 10: *Revised path from robot's new position to accommodate the discovery of the cart.*

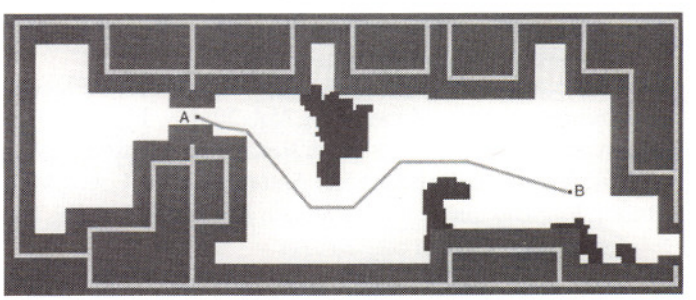

Figure 11: *Return path generated by the Planner avoids the previously discovered objects.*

errors. As the actual position and heading of the robot deviate from perceived values, the data entered into the absolute representation becomes increasingly inaccurate, until the model is eventually rendered useless. The ability to "flush" the map under these conditions by erasing all transient-object representations facilitates recovery, assuming some means is also provided to re-reference the robot. For more information on this subject, refer to *Sensors for Mobile Robots* (Everett, 1995a); dead-reckoning techniques and error sources are discussed in chapter 2, while the specifics of acoustical, optical, and RF position location techniques are presented in chapters 14-16. A very detailed treatment of more sophisticated navigational techniques is presented in *Navigating Mobile Robots* (Borenstein, et al., 1996).

About the Author

Commander H. R. (Bart) Everett, USN (Ret.), is the former Director of the Office of Robotics and Autonomous Systems at the Naval Sea Systems Command, Washington, DC. He currently serves as Technical Director for the tri-service Mobile Detection Assessment Response System (MDARS) robotic security program under development at the Naval Command Control and Ocean Surveillance Center, San Diego, CA. Active in the field of robotics research for over 20 years, with personal involvement in the development of 11 mobile systems, he has more than 80 technical papers and reports published and 19 related patents issued or pending. He serves on the Editorial Board for Robotics and Autonomous Systems magazine and on the Board of Directors for the International Service Robot Association, and is a member of Sigma Xi, the Institute of Electrical and Electronics Engineers (IEEE), and the Association for Unmanned Vehicle Systems International (AUVSI).

everett@nosc.mil

http://www.nosc.mil:80/robots/index.html

This article was adapted from "Modeling the Environment of a Mobile Security Robot," (Everett, et al., 1990).

References

Balch, T., "Grid-Based Navigation for Mobile Robots," The Robotics Practitioner, pp. 7-10, Winter, 1996.

Beckerman, M., Oblow, E.M., "Treatment of Systematic Errors in the Processing of Wide Angle Sonar Sensor Data for Robotic Navigation," Oak Ridge National Laboratory Technical Memo, CESAR-88/07, February, 1988.

Borenstein, J., Everett, H.R., Feng, L., Navigating Mobile Robots, ISBN 1-56881-058-X, A.K. Peters, Ltd., Wellesley, MA, 1996.

Brooks, R.A., "Solving the Find-Path Problem by Good Representation of Free Space," IEEE Transactions on System, Man, and Cybernetics, Vol. SMC-13, No. 3, 1983.

Brooks, R.A., "A Robust Layered Control System for a Mobile Robot," IEEE Journal of Robotics and Automation, Vol. RA-2, No. 1, pp. 14-20, 1986.

Brooks, R.A., Lozano-Perez, T., "A Subdivision Algorithm in Configuration Space for Findpath with Rotation," International Joint Conference on Artificial Intelligence, Karlsruhe, Germany, 1983.

Crowley, J.L., "Navigation for an Intelligent Mobile Robot," IEEE Journal of Robotics and Automation, Vol. RA-1, No. 1, March, 1985.

Everett, H.R., "A Multi-Element Ultyrasonic Ranging Array," Robotics Age, pp. 13-20, July, 1985.

Everett, H.R., Gilbreath, G.A., Tran, T.T., Nieusma, J.M., "Modeling the Environment of a Mobile Security Robot," NOSC Technical Document 1835, Naval Oceans Systems Center, San Diego, CA, June, 1990.

Everett, H.R., Sensors for Mobile Robots: Theory and Application, ISBN 1-56881-048-2, A.K. Peters, Ltd., Wellesley, MA, June, 1995a.

Everett, H.R., "Understanding Ultrasonic Ranging Sensors," The Robotics Practitioner, pp. 27-38, Fall, 1995b.

Everett, H.R., "Autonomous Navigation on a Shoestring Budget," The Robotics Practitioner, pp. 15-23, Winter, 1996.

Fryxell, R.C., "Navigation Planning Using Quadtrees," SPIE Mobile Robots II, Cambridge, MA, pp. 256-261, November, 1987.

Gilbreath, G.A., Everett, H.R., "Path Planning and Collision Avoidance for an Indoor Security Robot," SPIE Mobile Robots III, Cambridge, MA, pp. 19-27, Novemeber, 1988.

Harrington, J.J., Klarer, P.R., "SIR-1: An Autonomous Mobile Sentry Robot," Technical Report SAND87-1128, UC-15, Sandia National Laboratories, May, 1987.

Lee, C.Y., "An Algorithm for Path Connections and Its Applications," IRE Trans. Electron. Comp., Vol. EC-10, pp. 346-365, September, 1961.

Lozano-Perez, T., Wesley, M.A., "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," Communications of the ACM, Vol. 22, No. 10, pp. 560-570, 1979.

Moravec, H. P., Elfes, A., "High Resolution Maps from Wide Angle Sonar," Proceedings of the 1985 IEEE International Conference on Robotics and Automation, St. Louis, MO, pp. 116-121, March, 1985.

Rubin, F., "The Lee Path Connection Algorithm," IEEE Transactions on Computers, Vol. C-23, No. 9, pp. 907-914, September, 1974.

Winston, P.H., Artificial Intelligence, Addison-Wesley, Reading, MA, pp. 101-114, 1984.