Real-time Algorithms and Data Structures for Underwater Mapping

David N. Oskard[‡], Tsai-Hong Hong[‡], and Clifford A. Shaffer[‡‡]

[‡]Robot Systems Division
National Bureau of Standards
Gaithersburg, MD 20899

[‡‡]Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061

Abstract

As part of the Multiple Autonomous Underwater Vehicle (MAUV [†]) project at the National Bureau of Standards, a spatial mapping system has been developed to provide a model of the underwater environment suitable for autonomous navigation. The system is composed of multi-resolution depth maps designed to integrate sensor data with an *a priori* model, an object/attribute database for storing information about detected objects, and a set of flags to monitor abnormal or emergency conditions in the environment. This paper describes the structure of the mapping system and the algorithms used to map terrain and obstacles detected by acoustic sonar.

1. Introduction

In an autonomous system, the ability to predict, estimate, and evaluate the state of the environment largely determines the capabilities of the system. For the application of autonomous underwater vehicles, a means must be developed to model the world accurately enough to perform such tasks as obstacle avoidance, path planning, and long-range mission planning with sufficient speed to operate in real-time. Research at the National Bureau of Standards (NBS) has led to the development of a real-time control system (RCS) for multiple autonomous underwater vehicles (MAUVs) performing cooperative tasks [3]. Based on the theory of hierarchical control [1], the system is designed to perform both high and low-level planning tasks with a world model that provides timely information at multiple resolutions. The RCS is divided into several logical levels of complexity; at each level, the world model contains 6 modules:

[†] Funding for the MAUV project was provided by the Office of Naval Technology, Defense Advanced Research Projects Agency.

1) Update Module – interprets and adds incoming sensor data to the model,
2) Data Server – provides requested model information to the planner,
3) Sensor Prediction Module – provides expected sensor data estimates for use in validation by the sensor processor,
4) State Evaluator – monitors emergency conditions and the state of each RCS module,
5) Learning Module – categorizes information supplied by the database and other modules,
6) Database – for storage of information.

The emphasis of this paper is on describing the update module and portions of the database. For information on the other modules and the functionality of the world model, refer to [3].

In its current form, the MAUV world model mapping system consists of depth maps of lake bottom terrain, state variables, emergency flags, and a database of objects. We will focus primarily on the lake bottom and obstacle mapping functions of the world model and considerations in their design and implementation. The first section discusses the selection of data structures for representing the underwater environment, followed by a description of the mapping system. Algorithms used in the update module are presented, with an emphasis on the concept of confidence-based mapping. The discussion points out some of the strengths and weaknesses of the system and suggests directions for further research.

2. Criteria for Data Representations

The first and most important task in the development of an effective world model is choosing an appropriate data representation. In most mapping systems, a primary distinction can be made as to whether given information is spatial or non-spatial data. Depth maps or other models of a 3-dimensional environment are referred to as spatial data structures, whereas structures representing objects and their attributes are primarily non-spatial data structures. Although most objects in the world contain a spatial component, namely a location and 3-D structure in some frame of reference, a suitable representation for the associated descriptive information is an object/attribute database. A simple object/attribute database has been developed to store data of this nature. Each object structure contains an identification number along with a 32-bit mask describing known capabilities. A linked list of attributes stores sensed characteristics for each object (e.g. X, Y, Z coordinates, velocity vectors, etc.) with the ability to store multiple entries over time in a circular queue. It is flexible enough to allow 3-D representations of objects to be stored as attributes themselves by storing pointers to the 3-D data structures in the attribute list.

Choosing a data representation to model the underwater environment is a more diffi-

cult problem. One solution is to implement a complete 3-D data structure to model the lake bottom and all detected obstacles. An example of such a representation is the octree [5,7], which considers the working universe to be one large cubic region. The universe is recursively divided into 8 equal cubic sub-regions or "octants" until each sub-region is considered to be homogeneous, that is, either completely empty space or completely full, containing an object or the lake bottom. This approach has been used successfully to represent an automated factory environment [4]. However, in the application of autonomous underwater vehicles, an octree representation cannot keep up with the demands of the real-time environment when CPU resources must be shared by more than one RCS module. The cost of bookkeeping is high with octrees, as each new sonar reading may cause the tree to decompose or merge several nodes. With each decomposition, 8 new nodes must be allocated for the newly created subregions, and every merge operation requires the reclamation of unused nodes. Considering that the CPU is shared and the sampling rate for the six sonar sensors on a prototype vehicle is 0.6 seconds [6], a full 3-D representation was not attempted.

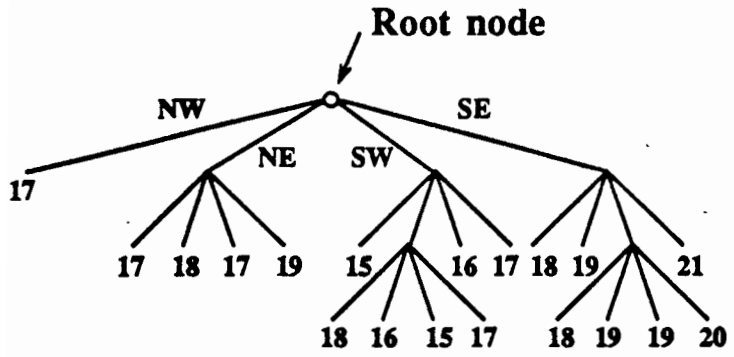
3. Global and Local Map Configuration

A less computationally intensive approach is to use a $2^1/_2$-D representation, storing only depth information for any map location. The current autonomous underwater vehicle (AUV) world model uses two types of $2^1/_2$-D data structures for its mapping scheme: a set of global maps, each of which contains data for the vehicle's operational domain, and region-of-interest maps, which only store a localized area around the AUV's current location. The *region quadtree* [14,15,16], which is the 2-dimensional analogue of an octree, is used to represent depth maps at a global level. The root node of the tree represents a 2-D square area that is recursively decomposed into 4 quadrants, sub-quadrants, and so on, until each node contains only one depth value. Because each branch of the tree contains half as many nodes as an octree branch, quadtree updates are generally faster than octree updates and the structure consumes less storage as well. As with octrees, a node is only further decomposed if it is not homogeneous. This property leads to storage savings over array data structures when the data is sparse, or when the data set contains large homogeneous regions such as islands or flat regions of a lake bottom [see Figure 1]. The quadtree approach also allows point and linear feature data to be represented using compatible data structures as described below.

3.1. Global Quadtree Implementation

The quadtree representation used for this project is a compact implementation of a *pointer-based* quadtree. The pointer-based quadtree explicitly stores pointers from each internal node *I* to *I*'s four children. This is in contrast to the *linear* quadtree which stores a list of leaf node values sorted by block location [2]. An explicit pointer structure is used because the quadtrees and other structures are maintained in memory when in use.

Our implementation reduces storage requirements over the common pointer-based



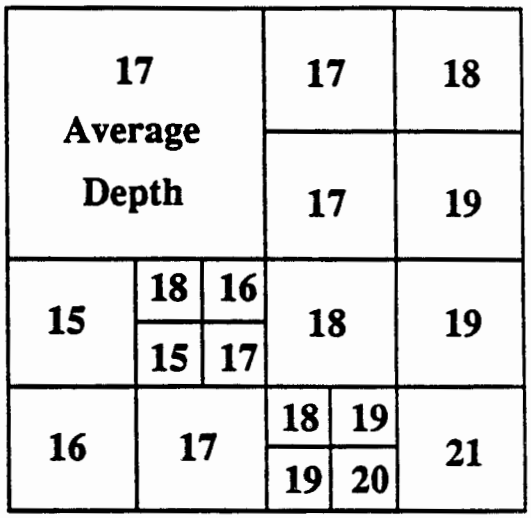


Figure 1. A region quadtree representation for a typical depth map. Only nodes containing more than one depth value are further decomposed. Each branch of the tree correponds to northwest, northeast, southwest, and southeast quadrants of a given region (shown in left to right order).

implementation which stores four pointer fields and a value field with each node of the tree. Instead, we only store pointer fields with internal nodes, commonly referred to as *GRAY* nodes. A GRAY node G is represented by a block of four contiguous descriptor fields, each corresponding to one of G's children. In the case where a child is itself a GRAY node, the value stored in the descriptor is a pointer to the block of storage representing that child. In the case where a child is a leaf node, the value stored is simply the depth value for the block. This representation is illustrated by Figure 2 which shows the storage structure for the quadtree of Figure 1. In Figure 2, each of the four child fields has an extra bit field associated with it to distinguish between GRAY children (a '0' bit) and leaf children (a '1' bit). In actual implementation in the C programming language on Motorola 68020 processors, GRAY nodes are represented by four contiguous 32-bit words. The low order bit of each 32-bit val-

	Node ID	NW		NE		SW		SE	
Root node →	1	17	1	→ 2	0	→ 3	0	→ 4	0
	2	17	1	18	1	17	1	19	1
	3	15	1	→ 5	0	16	1	17	1
	4	18	1	19	1	→ 6	0	21	1
	5	18	1	16	1	15	1	17	1
	6	18	1	19	1	19	1	20	1

Figure 2. Illustration of quadtree implementation. Each GRAY node has four fields, labelled NW, NE, SW, and SE. Each field has 2 subfields: the value and the GRAY/Child descriminator.

ue distinguishes between a GRAY child and a leaf child.

This implementation avoids the wasted storage of four null pointers associated with each leaf node to indicate that no children are present. The data structure stores a single pointer to each GRAY node and a value for each leaf node (i.e., no explicit pointers to leaf nodes are stored). Thus the total storage required is a single 32-bit value for each node in the tree for both leaf nodes and GRAY nodes. Since a quadtree of L leaves requires $\frac{4L-1}{3}$ nodes, this is also the number of 32-bit words required.

Three region quadtrees are used to represent different aspects of the global depth maps. One of these is the *a priori* map, which contains data from a survey of the lake bottom and does not change during the mission. A priori data was collected for an approximate area of 1.6 km^2 in Lake Winnapesaukee, New Hampshire (the 1987 test site). The resulting set of irregularly spaced data points was mapped to a regular grid using Renka and Cline's triangulation algorithm [12,13] and converted to quadtree format using a region quadtree construction algorithm [17]. The a priori quadtree in this instance does not yield much storage savings over a grid structure, as the lake survey data is of low resolution and could be represented by an array smaller than 512 by 512; however, using a quadtree provides a convenient means for integrating a priori map information with incoming sensor information.

A separate *sensor* quadtree is used to store higher resolution depth values collected from the AUV sonar sensors during vehicle test runs. Both downward- and forward-looking (obstacle avoidance) sonars are used in refining the sensor map, which is overlaid onto the a priori data to give the most accurate view of the world. A third quadtree stores a depth confidence value for each node in the tree. The confidence map supports the important function of distinguishing spurious sonar readings caused by debris or signal inconsistencies from actual obstacles that must be detected and avoided. Additional bit fields in the confidence map node value are used as flags to mark the vehicle's track through the water, as well as to record whether a given node has been updated with downward or forward-looking sonar. This extra information could be useful in classifying newly discovered objects and in examining the AUV's complete path after a mission. The region quadtree is particularly efficient for sensor and confidence map representation, since unexplored portions of those maps are empty. Such areas can be represented by a small number of nodes in the tree.

3.2. Point and Linear Feature Representation

In addition to the quadtrees used to represent region data, point and line storing quadtrees have been implemented to provide locations of known objects and topographic features of the lake bottom used in high-level path planning. This simplifies tasks such as locating the nearest other vehicle to a given location or plotting a course along linear topographic features like ravines or underwater pipelines. The planner can then retrieve information in a region of interest without having to search the object/attribute database directly. The global maps provide an efficient method for storing large quantities of data while maintaining the spatial relationships between different types of features.

Point objects are represented by means of a *point region* (PR) quadtree [14]. The PR quadtree is similar to the region quadtree except that its decomposition rule is based on the number of point objects located within a block. If a block contains no points or a single point, then this block is represented by a single leaf node. Otherwise, the block is decomposed into quadrants and sub-quadrants recursively, until each block contains at most a single point. Implementation of the PR quadtree is identical to that of the region quadtree illustrated in Figure 1a. Each point object is identified in the object/attribute database by means of an index number. Leaf nodes of the PR quadtree store this index. Operations such as finding the nearest object to a specified location and locating all objects within a specified distance of a location have been implemented.

Linear features are represented by means of the *PMR* (or *point matrix random*) quadtree [9]. Upon insertion of a line segment into a block B, the PMR quadtree decomposes B exactly once if the number of line segments already contained within B exceeds some threshold (in our case, the threshold is four segments). This does not guarantee that a block will contain four or fewer segments, since segments lying very close together may pass through the same block even after decomposition; however blocks rarely contain much more than four segments. The main objective of this decomposition rule is to reduce the number of line objects contained within a single block so as to make processing efficient. Note that varying numbers of line segments may be associated with a single block. Our PMR quadtree implementation is identical in structure to that of the region and PR quadtree implementations, except that leaf nodes store a pointer to a linked list of the line segments contained within that node. Operations such as finding the nearest line segment to a point have been implemented. The PMR quadtree has yet to be integrated into the map representation and planning portions of the MAUV system. However, availability of the PMR quadtree implementation will enable future versions of the planner to operate directly on the higher level representation of ridge and trench lines rather than on the low level data currently provided by the sensor and confidence maps.

3.3. Local Maps

The AUV world model makes use of grid or array data structures for storing local region-of-interest maps of varying resolutions. Arrays are used for their fast, constant access and update time and for ease of implementation. Local maps are generated from the global quadtree database, first by extracting a priori map data for the region, then overlaying the data stored in the sensor and confidence quadtrees, which are presumed to be more accurate than the lake survey information. In fusing the three sets of data, all three quadtrees are traversed over the local map region. The value stored in the confidence quadtree for any map location is used to determine whether or not there is sensor data for that area. Because the updating algorithm only stores data in the sensor quadtree if it has a confidence measure above the level assigned to a priori data, any node for which there is sensor data uses the sensed value. The local map uses a priori knowledge only if insufficient sensor data has

been collected for that node. Confidence quadtree values are also copied into the local map to be used extensively in the updating algorithm.

Each map has an offset coordinate which defines its position relative to a global origin. As the vehicle moves to within a predefined distance of the current local map's edge (in this case 64 meters defined as 1/4 the width of the map), a new shifted map is generated in a second buffer so as to keep the AUV nominally centered. Before this is done, any newly acquired sensor and confidence data in the old local map must be written to the respective quadtrees, making it available for the new map. During the generation process, incoming sonar data is used to update the old buffer (still being used by the planner), but is also held in a data queue to be applied to the new buffer after it has been extracted. When the new map has been completed, the software simply swaps the two map pointers and updates the global coordinate offsets of the new local map. With this double buffering arrangement, the planner can still access current data while a new map is being generated. This provides the capability for asynchronous world model queries as required in the AUV's multiprocessing environment.

The algorithm that copies data from the quadtree representing the global map to the local map array is a simple directed traversal of the quadtree beginning at the root node. If the root is a leaf node, then the value of the root is copied to the portion of the array overlapped by the root. If the root is a GRAY node, the algorithm is recursively called for only those children of the root that overlap the local map. Thus, only that portion of the global quadtree overlapping the local map is processed.

Updates of a global quadtree from the local map take place when the AUV moves beyond the center region of the local map. The quadtree update algorithm is analogous to the array update algorithm, except that one can view the quadtree update algorithm as being driven by a traversal of the array rather than the quadtree. Each pixel of the array is visited not in raster scan order, but in the order that the pixels would be visited by a preorder traversal of the corresponding quadtree. The pixel addresses for such a traversal are calculated by interleaving the bit representation of the x and y coordinates for each pixel, and visiting the pixels in ascending order; this is known as Morton order, N-order, or Z-order [2]. As each local map pixel is visited, it is compared against the corresponding node in the global quadtree, whose node value is updated if required. Processing in Morton order ensures that all pixels of the current quadtree node will be checked before the next node must be located.

3.4. The Mapping Hierarchy

As mentioned in section 1, the RCS is a hierarchical control system, meaning that tasks to be performed are functionally decomposed into distinct levels of complexity. Various levels of the control hierarchy require different local map resolutions. Different types of data may be needed at each level. Generally, the resolution of maps at each level differs by an order of magnitude or more. All local maps are implemented as array data structures and only the lowest level (highest resolution) local map updates the global quadtrees. Figure 3

shows the mapping hierarchy for a generalized data set. In the current implementation, the mission level map divides the area into a coarse grid of approximately 25 x 25 picture elements or *pixels* per square mile, each pixel storing the average depth of the corresponding area, along with probabilities of detection and destruction of the AUV at each location based only on a priori knowledge. It functions primarily as a guide in tactical planning.

The next two levels in the hierarchy, the group and vehicle levels respectively, share the same local map for this data set. Each pixel of the local map stores the minimum and maximum known depths over a 4 x 4 meter area. It serves the purpose of providing information for high level navigation tasks, such as determining the probability that an area is traversable by one or more vehicles. This map is updated as new information is added to the lowest level map, known as the elemental move or "E-move" level map.

The E-move local map has the highest resolution, and is used in determining the traversability of a path between two specified points. The world model returns a probability that the path is traversable based on the information in this map and the needs of the plan-

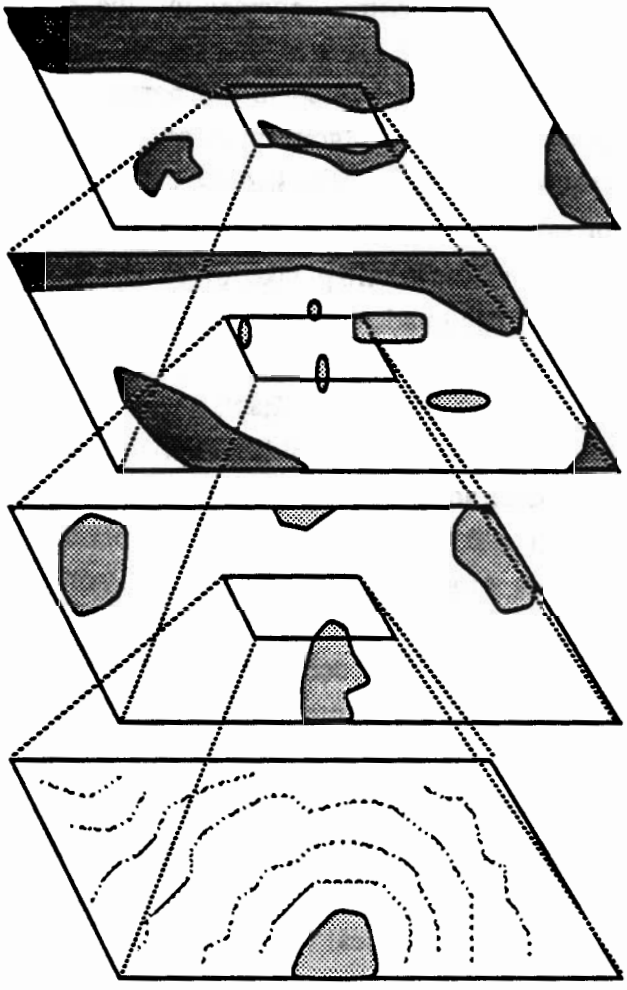


Figure 3. The MAUV mapping hierarchy.

ner. For example, the output may be a percentage of pixels for which the AUV clears the lake bottom over the hypothesized path. In the simplest case, the world model can provide a probability of 1 if all of the pixels are traversable, or 0 if any are obstructed. Typically, the pilot planner will query the world model for the traversability of several paths, using a modified A* search algorithm [11] to choose the safest, most efficient path available. The E-move map is also the level updated directly by sensor readings; its modifications are propagated up through the hierarchy.

In addition to the world model maps, a set of emergency flags are constantly monitored for conditions which require immediate action. These flags are raised by the lowest level of the sensory processing module when abnormalities are detected in sensor data. Each bit in the emergency flag register corresponds to a different condition, such as low fuel, water leakage, sensor failure, etc. One bit is reserved for an imminent collision condition, meaning that the system has not responded quickly enough to avoid an obstacle, an oncoming ship, or possibly even another AUV. This guards against the inherent delay in detecting obstacles, adding them to bottom maps, and planning paths around them. It also provides a backup safety mechanism for the case that a bug in the software has caused an object not to be detected. While the mapping system is designed for use in dynamic path planning, its performance in terms of object detection delay is ultimately tied to sensor cycle times, CPU efficiency, and resource allocation, in addition to the algorithm design and implementation.

4. The Mapping System Algorithm

For the first implementation of the mapping system, the confidence assignment algorithm has been greatly simplified. A complete implementation would include a stochastic analysis of the sonar data, taking into account the increased likelihood of error as the range to the target increases, as well as uncertainties in the AUV's navigational data [18]. Here we have assumed that the incoming data has been filtered by a sensory processing module to minimize such inaccuracies.

At the beginning of a mission, the AUV initializes the global and local maps, reading available a priori knowledge from a secondary storage device. In our prototype, the lake bottom survey map is read from an optical disk storage medium into the a priori quadtree. The sensor quadtree is initially composed of a single, empty node, though it could also contain sensor data stored from previous missions if available. Likewise, the confidence quadtree is initialized as a single node containing a base confidence value, unless there is confidence data from a previous mission. In general, the AUV world model starts up in a state of total dependence on a priori knowledge, gradually becoming more reliant on the current sensor map as data is collected.

4.1. Confidence-based Mapping

One principle that has become central to the map updating algorithm is the concept of confidence-based mapping. By modifying values in the confidence map, the interpretation of incoming sensor data can be controlled, which in turn influences the vehicle's behavior. Con-

fidence values are incremented or decremented from an initially assigned base value as confirming or conflicting information is received. The range of values is set by the user at compile time, the default being from 0 to 20, where a value of 20 corresponds to 100% confidence. The upper bound on a pixel's confidence guards against the problem of not detecting a moving object due to overconfidence in data which no longer is accurate. When the system receives conflicting sensor data, the local map update module waits until a pixel's confidence drops below a threshold before changing its depth value. The threshold value is a prespecified constant determined by the maximum change a single sensor reading can affect, though this can also be assigned by the user.

The other constants in the algorithm are the base confidence values. One is assigned to a priori data; the other, to pixels whose depth values have just been modified with sensor information. These constants directly affect the behavior of the mapping system by determining the amount of conflicting sensor data required to cause a change in the depth map. If the base confidence given to the a priori map is relatively high, it will take several conflicting sonar readings before a pixel's depth is updated with new sonar information. The higher the a priori confidence, the longer the AUV will take to "see" and subsequently avoid a new obstacle. On the other hand, higher initial confidence values reduce the effects of erroneous sensor readings.

Another feature of the confidence algorithm is the ability to weight the input of each type of sensor differently. In our current implementation, depth sonar readings are able to modify a pixel's confidence by 10% per reading, while obstacle avoidance sonars only modify confidence values by 5% per reading. Although these particular values are somewhat arbitrary--based on the assumption that depth sonar is more accurate than obstacle avoidance sonar due to fluctuations in the AUV's orientation--the basic idea has been to create a mapping system whose behavior can be modified by changing its parameters, without requiring substantial effort on the part of the programmer.

4.2. Evaluating Sonar Data

When presented with new information, a world model may classify it into one of three categories. The new data may affirm knowledge already in the model, causing the confidence in that knowledge to be increased; it may present conflicting information, in which case an intelligent world model would decrease its confidence in that knowledge; or the incoming data may be classified as irrelevant, requiring no action. Our ability to function in the world as humans depends heavily on our ability to make such judgements. The same is true in autonomous navigation. In the AUV prototype, inputs are limited to two types of range data: downward-looking sonar and forward-looking obstacle avoidance sonar. Because of this simplification, data classification of this sort is fairly straightforward.

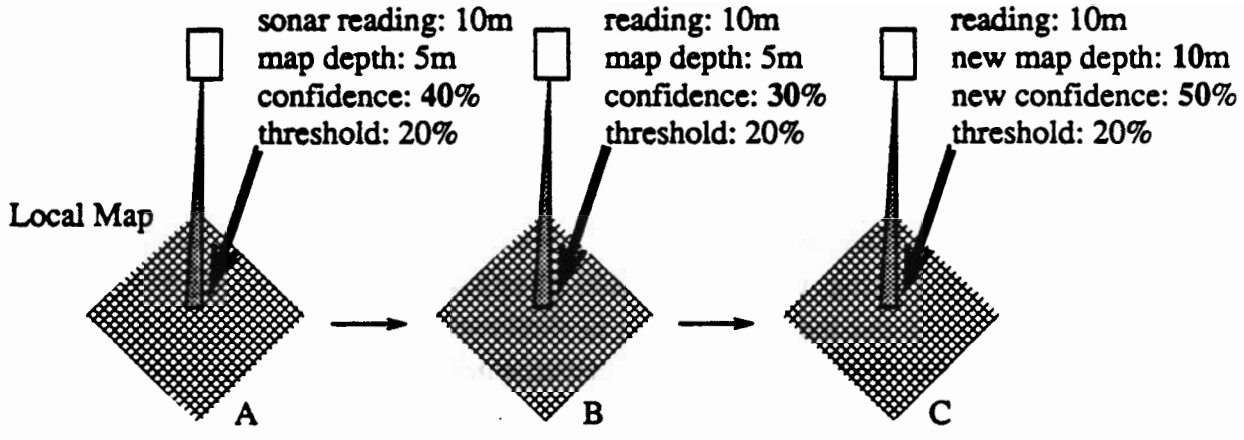
Although the return from a sonar source can be somewhat eccentric [8], a sonar reading is assumed to represent a conic beam of approximately six degrees in diameter, with a coordinate transformation describing its position with respect to the vehicle. The map updat-

ing routine currently handles each sensor separately, using different approaches for the downward and obstacle avoidance data to modify the local map. As a first step in processing, a sonar beam is projected into the X-Y plane, transforming its coordinates with respect to the vehicle into world model coordinates with respect to the global origin.

4.2.1. Depth Sonar

In updating the map from the downward-looking sonar, the algorithm first computes an approximate neighborhood size of pixels to be updated around the current AUV location depending on the width of the beam and the distance to the lake bottom. This approach is possible because the pitch and roll properties of the vehicles are negligible, causing the depth sonar to always point straight downward. Given that the beam width of the sonar is fixed and the range is returned by the sensor, a closed-form trigonometric solution can be performed using a lookup table. Although the 2-D projection would be best represented as a circular region, for our purposes, a square neighborhood is sufficiently accurate and more efficient to update. The depth stored at each pixel of the neighborhood in the local map is compared to the observed sonar reading. If the two values are not within an acceptable margin of error, the conflicting data causes the pixel's confidence to be lowered. If the two depth values are in agreement, the confidence value is incremented unless it has already reached the maximum allowed. Whenever a pixel's confidence value drops below the predefined threshold, it takes on the new depth reading and is assigned a base confidence value [see Figure 4]. For the depth sonar, all information is classifiable as either conflicting or agreeing with the knowledge already in the model. None of the data is irrelevant in this case.

Figure 4. Three stages of a map update from depth sonar: (A) Incoming sonar data conflicts with the world model (B) Confidence values for the corresponding world model pixels are decremented (depth is not modified yet) (C) After repeated conflicting readings, confidence drops below the threshold, the depth is updated, and a new confidence assigned.



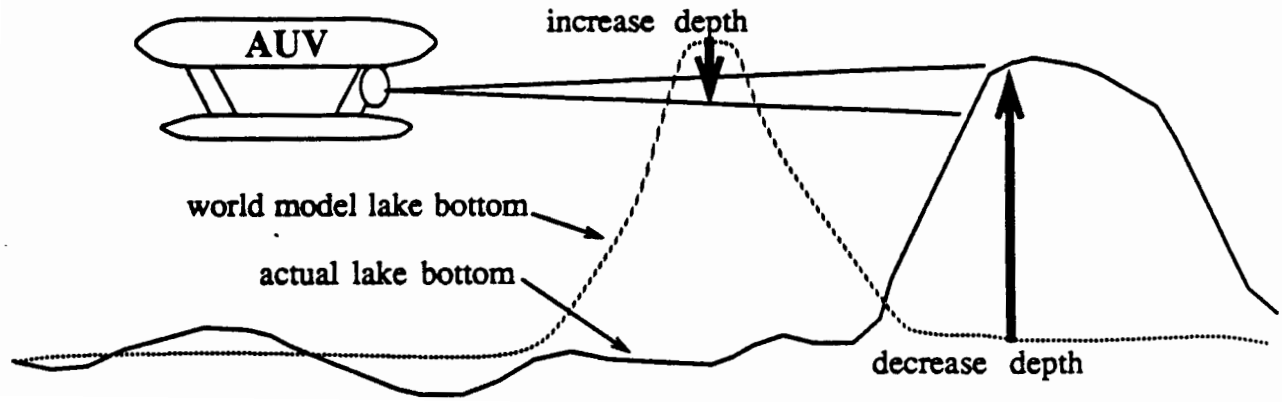


Figure 5. Updates from obstacle avoidance sensors remove false obstacles in the world model by increasing depth values in the map. Obstacles are added by decreasing the depth, in effect, raising the bottom of the lake model.

4.2.2. Obstacle Avoidance Sonar

The obstacle avoidance mapping algorithm is more complicated. Here the projection of the cone into the two-dimensional plane approximates a triangular region. The cone itself is approximated by two planar surfaces representing the top and bottom surfaces of the cone, calculated at three degrees above and below the central axis of the cone respectively. Due to the relatively coarse resolution used in the obstacle avoidance algorithm ($0.5m^3$ per pixel) and the narrow width of a sonar beam, this does not introduce significant error into the calculations. As with the depth sonar algorithm, each pixel in the 2-D projection is examined and updated if its confidence value drops below the threshold. Forward-looking sonar readings provide two types of information: a given pixel may be clear, or it may be obstructed by an obstacle. When the AUV detects an obstacle, the mapping algorithm adds the information to the local map by raising the modeled bottom of the lake at that location (i.e. making it shallower) [see Figure 5]. It is also an essential function of the world model to be able to remove hypothesized obstacles in the local map as well as add them. For each pixel in the triangular projection, if the three dimensional distance (measured along the sonar cone trajectory) from the sonar source to the current pixel being examined is less than the range returned by the sensor, the pixel is assumed to be clear. No obstacle was detected there, so the depth at that location in the local map should reflect this information. Its value should be greater than or equal to the depth of the bottom surface of the sonar cone at that location, since any object obstructing the beam would presumably cause the sensor to return the range to that object. If the local map value is shallower than the beam, it conflicts with the new sensor data and the confidence value is decremented. If this results in a confidence lower than the threshold, the pixel is reassigned the depth value of the bottom surface of the

cone and a new base confidence value. Note however that a local map value in agreement with sonar information does not necessarily increase its confidence. The sonar beam may be projected in front of the vehicle when the AUV is near the surface, and a clear reading near the surface would not yield any information about the depth of the lake bottom if we already have some a priori knowledge that the lake is approximately N meters deep. In this case it would be considered irrelevant data.

The same is not true for pixels in the projection whose distance from the sonar source is greater than or equal to the range returned by the sensor. These pixels correspond to detected obstacles and the depth values in the local map are compared to the top surface of the cone.† Here the local map data should be at least as shallow as the top surface of the beam to be in agreement with the sensor reading. If the map data does agree, it represents confirmation of an existing obstacle and the confidence value should be incremented. It should be noted that this confirmation only supports the hypothesis that there is an obstacle at the depth it was detected; no conclusions can be drawn as to the true height of the object or whether it extends all the way to the lake bottom. Although this may lead to some innacuracies in the depth map, an array of multiple sensors pointing in different directions minimizes this effect.

In a similar manner, if the model continually disagrees with the sensor reading, the confidence is decremented until the depth value is reassigned to the depth of the top surface of the cone, making the model shallower. Its confidence is again initialized to a base value.

5. Mapping System Performance

Processing time for a sonar input is roughly proportional to the number of pixels that must be examined in the depth map. On the software development system used at NBS, a Sun-3 workstation, obstacle avoidance processing ranges in speed from 30 milliseconds of CPU time for short distances of approximately 10 meters, to 100 milliseconds for the maximum accepted reading of 50 meters. Depth sonar updates are practically instantaneous, due to the lack of computation required. Since the sensor cycle time is approximately 0.6 seconds and the vehicles used are equipped with an array of 5 obstacle avoidance sensors [6], it is not possible to process each sonar reading separately while sharing the CPU with several other processes. To improve performance, a preprocessing filter has been added to the front end of the system that combines similar sonar readings from the same sensor into a single, more heavily weighted reading. The filter passes the average range returned by the sensor, the average position of the vehicle, and the number of combined readings to the mapping system. Tolerances for what constitute similar readings are parameters to the system. Only the confidence assignment routine must be modified to accomodate the filtered data, multiplying the weighting factor for the current sensor by the number of combined readings. This filtering process enables the system to operate well within hardware timing constraints.

† One detail worth mentioning is that the two-dimensional projection of the cone is extended by two pixels so that it will not only include the projection of the beam, but also a portion of the detected obstacle.

Another time-consuming operation is the transfer of data between local maps and global maps. Each time the AUV crosses the boundary of the current local map (really 64m from the map's edge as described in section 3.3), the world model must examine the entire array for pixels which have been updated, modify the sensor quadtree, and extract the new local map. This process takes on the order of several CPU seconds. For this reason, a separate process should peform this task in the "background" while processing of the sensor data continues using the old map buffer. As previously mentioned, data collected while the new window is being extracted must be stored in a queue to be applied to the new local map. Although this feature has not been fully implemented yet, it is expected that the queued data will not cause the system to fall behind in its processing if filter parameters are set properly. If necessary, the algorithm can be further modified to only store readings which caused the old local map to be updated.

6. Discussion

The MAUV mapping system has been developed to provide an effective means for autonomous navigation. During the first year of the project, much work has been done on the problem of short term planning for obstacle avoidance. The $2\frac{1}{2}$-D depth maps using the confidence-based mapping algorithm have proven to be an adequate solution. A dynamic planner continuously queries the world model for the probability of safe traversal over the planned trajectory and is able to modify the vehicle's path to avoid obstacles as soon as they are detected. It typically takes 3-4 supported readings from the obstacle avoidance sonar before the map is modified to show the obstacle, though this property can be adjusted by altering the confidence parameters of the system. For example, a low base confidence value would make the mapping system more responsive to sonar readings, decreasing obstacle detection time, but increasing the probability of error due to sensor noise.

Because of the $2\frac{1}{2}$-D nature of the data structures, the current implementation is unable to model underwater formations such as caves and ledges. All obstacles are modeled as elevated portions of the lake bottom. With this representation, the path planning module does not consider routing alternatives which would go under detected obstacles. The system also does not yet include facilities for representing the 3-D shape of objects; however, the locations of objects can be stored in the PR quadtree with associated 3-D data maintained in the object/attribute database. Because the database can be configured to store arbitrary parameters for each data type, descriptive information such as volume, centroid location, and number of holes may accompany an entry in the PR quadtree. As mentioned earlier, one of the entries in the database may even be a pointer to an extensive 3-D representation of the object.

The next generation mapping system will include the full integration of the PMR quadtree with the planning modules to store route and topographic information. This will enable high level planners to follow contours of the lake bottom and find paths of minimal risk to the vehicles. Additional sensor data from laser ranging devices and underwater cameras

for inspection of objects may be included as well. Another topic for future research is the use of spherical representations of the underwater environment in real-time world modeling. Much more work is needed to fully test and develop the system, but progress so far indicates that this approach will produce a fast and reliable system for real-time underwater mapping.

7. Acknowledgments

The authors would like to thank Dr. Tsung-Ming Tsai and Shujen Chang for their contributions to this work.

References

[1] J. Albus, "Hierarchical Control for Robots and Teleoperators," *IEEE Workshop on Intelligent Control, Albany*, NY, August 26-27, 1985.

[2] I. Gargantini, ''An Effective Way to Represent Quadtrees'', *Communications of the ACM 25*, December 1982, 905-910.

[3] M. Herman and J. Albus, "Overview of the Multiple Autonomous Underwater Vehicle (MAUV) Project," *IEEE International Conference on Robotics and Automation*, April 1988.

[4] T-H Hong and M. O. Shneier, "Describing a Robot's Workspace Using a Sequence of Views From a Moving Camera," *IEEE Trans. Pattern Analysis and Machine Intelligence*, 1985.

[5] C. L. Jackins and S. L. Tanimoto, "Oct-trees and Their Use in Representing Three-dimensional Objects," *Computer Graphics and Image Processing*, 1980, pp. 249-270.

[6] J. Jalbert, "Low Level Architecture For the New EAVE Vehicle," *Fifth International Symposium on Unmanned Untethered Submersible Technology*, June, 1987, Vol. 1, pp. 238-244.

[7] D. Meagher, "Geometric Modeling Using Octree Encoding," *Computer Graphics and Image Processing*, 1982, pp. 129-147.

[8] H. P. Moravec and A. E. Elfes, "High Resolution Maps from Wide Angle Sonar," Proceedings IEEE Conference on Robotics and Automation, March 1985, 116-121.

[9] R.C. Nelson and H. Samet, ''A Consistent Hierarchical Representation for Vector Data'', *Proceedings of the SIGGRAPH'86 Conference*, Dallas, August 1986, 197-206.

[10] W. M. Newman and R. F. Sproull, Principles of Interactive Computer Graphics, 2nd ed., McGraw-Hill, New York, 1979.

[11] N. J. Nilsson, Problem-Solving Methods in Artificial Intelligence, McGraw-Hill, New York, 1971.

[12] D. J. Orser and M. Roche, "The Extraction of Topographic Features in Support of Autonomous Underwater Vehicle Navigation," *Fifth International Symposium on Unmanned Untethered Submersible Technology*, June, 1987, Vol. 2, pp. 502-514.

[13] R. J. Renka and A. K. Cline, "A Triangle-Based C^1 Interpolation Method," *Rocky Mountain Journal of Mathematics*, Vol. 14, No. 1, Winter 1984, pp. 223-237.

[14] H. Samet, "The Quadtree and Related Hierarchical Data Structures," *ACM Computing Surveys 16*, June 1984, pp. 187-260.

[15] H. Samet and R. E. Webber, "Storing A Collection of Polygons Using Quadtrees," *ACM Transactions on Graphics 4*, 1985.

[16] H. Samet, A. Rosenfeld, C.A. Shaffer, and R.E. Webber, "A Geographic Information System Using Quadtrees," *Pattern Recognition*, Nov./Dec., 1984, 647-656.

[17] C.A. Shaffer and H. Samet, "Optimal Quadtree Construction Algorithms", *Computer Vision, Graphics, and Image Processing*, March 1987, 402-419.

[18] W. K. Stewart, "A Model-Based Approach to 3-D Imaging and Mapping Underwater," *Proc. ASME Conference on Offshore Mechanics and Arctic Engineering*, Houston, TX., Feb. 1988.