Proceedings of the Third International Symposium on Intelligent Control, Arlington, VA, August 1988.

Planning and World Modeling for Autonomous Undersea Vehicles

Martin Herman, Tsai-Hong Hong, Scott Swetz, David Oskard

Robot Systems Division
National Bureau of Standards
Gaithersburg, MD 20899

Mark Rosol

Intelligent Automation, Inc.
1715 Glastonberry Rd.
Rockville, MD 20854

ABSTRACT

The goal of the Multiple Autonomous Undersea Vehicles (MAUV) project is to have multiple undersea vehicles exhibiting intelligent, autonomous, cooperative behavior. The MAUV control system is hierarchically structured and incorporates sensing, world modeling, planning and execution. This paper describes the overall architecture and then focuses on the planning and world modeling components required for navigation of the vehicles.

1. Introduction

The NBS Multiple Autonomous Undersea Vehicles (MAUV) project has, as its objective, the demonstration of multiple undersea vehicles exhibiting intelligent, autonomous, cooperative behavior. The project involves the development of a real-time intelligent control system that performs sensing, world modeling, planning and execution.

The MAUV control system is hierarchically structured [1, 2] and is divided into three main components. These are *sensory processing, world modeling*, and *task decomposition*. The goal of the task decomposition component is to perform real-time decomposition of task goals by means of real-time planning, execution and task monitoring. The world modeling component performs the following functions: (a) it maintains a central real-time database of information about the state of the world and the internal state of the system, (b) it updates this database with information from sensory processing, (c) it provides expectations of incoming sensory data, (d) it responds to queries from the task decomposition component based on information in the database and on evaluations of possible future states of the world. The goal of the sensory processing component is to detect and recognize patterns, events and objects, and to filter and integrate sensory information over space and time.

The world model serves as a buffer between the sensory processing component and the planners and executors of the task decomposition component. That is, queries about the world required for planning and execution are made to the world model, and sensory processing provides information that is used to update this world model.

The control system is divided hierarchically into several levels. We view this kind of hierarchical division as a means of converting broad, high-level goals into commands to actuators, motors, communication transducers, sonar transducers, etc.

In the task decomposition hierarchy, the highest level, the *mission* level, converts a commanded mission into commands to each of a set of groups of vehicles. These commands involve tasks that treat a whole group of vehicles as a single unit. The *group* level converts group commands into commands to each of the vehicles in the group. These commands involve large tasks for each vehicle. The *vehicle task* level converts task commands into elemental moves and actions for the vehicle. The *e-move* (elemental move) level converts elemental moves and actions into intermediate poses. These are converted into smooth trajectory positions, velocities, and accelerations by the *primitive* level. Finally, the *servo* level converts these into signals to actuators, transducers, etc. The MAUV control system is based on the one developed for the Automated Manufacturing Research Facility at NBS [5].

This paper focuses on the planning and world modeling components at the vehicle task and e-move levels.

2. Hierarchical Planning and Execution

Before describing the elements of hierarchical planning and execution, we will provide our working definition of a plan, and describe the difference between planning and execution. A plan is made up of actions and events. The events are either events in the world or events in the internal state of the system. We represent a plan as a graph. The nodes of the graph represent actions and the arcs represent events. The purpose of the planner is to obtain a plan graph. It can either generate it or retrieve it from a database.

We define execution as the process of carrying out a plan. The purpose of the executor is therefore to step through the plan graph. When the executor arrives at a node of the plan graph, it "executes" the action associated with the node. If an action is at the lowest level of the hierarchy, then executing it involves sending signals to hardware. Otherwise, executing an action involves sending it to a lower level where it can be decomposed. As the executor sits at a node of the plan graph, it monitors for events associated with arcs leading out of the node. This monitoring is done at a fast cycle rate. The process of monitoring for an event consists of querying the world model database for that event. If an event has occurred, the executor follows the arc corresponding to that event and steps to the next action.

The notion of hierarchical planning is the following. An action is first input to the top level as a task command. This task is decomposed both spatially and temporally. Spatial decomposition means dividing a task into logically distinct jobs for distinct subsystems. For example, the group level will have a different planner for each vehicle in the group. Temporal decomposition means decomposing a task into a sequence of subtasks. The first step in the plan is then the input task to the next lower level, and this, in turn, is decomposed both spatially and temporally. At each successively lower level, the actions become more detailed and fine structured.

At each level of this hierarchy, the input task to the level first goes to a Planner Manager. The Planner Manager performs spatial decomposition by assigning jobs to each of a set of planners. The Planner Manager also coordinates planning among these planners. The planners, operating in parallel, generate their respective plans. Associated with each planner is a separate executor which executes the plan. The executors also operate in parallel.

3. Planning

Unlike other systems that replan for dynamic environments only when a specific change has been detected, each planner is constantly replanning and, thus, continually updating its view of the world and its solution to the search problem. Figure 1 illustrates the basic components of the planning system at each level of the hierarchy. In order of execution, the system functions as follows.

3.1. Planner Manager

The Planner Manager receives as input a command set consisting of the command itself, a set of command parameters, a set of priorities and a set of command constraints. The command parameters instantiate the command. For example the command GO_STRAIGHT is instantiated by goal point coordinates x, y and z. The priorities are values indicating the importance of stealth, destruction, time, and energy. Command constraints indicate planning bounds. For example to avoid obstacles, the planner may have to obey the constraint of not deviating more than a certain distance from a straight line path between start and goal point.

Upon receipt of the command, the Job Assignment Module of the Planner Manager partitions a command into a set of functionally distinct tasks to be performed by different subsystems. Presently, it uses a table lookup architecture to do this. As an example, an input command to the vehicle task level is partitioned into 3 distinct subtasks, one each for navigation, communications, and sensing. Output from the Job Assignment Module is in the form of individual commands, one for each planner.

If a conflict or inability to plan within the given constraints and world state is encountered by any of the individual planners, it is reported to the Plan Coordination Module of the Planner Manager. The Plan Coordination Module reviews the situation, revises the command, and resends it to each of the planners. If the Plan Coordination Module cannot revise the plan, it reports this condition to a higher-level planner which attempts to rectify the situation similarly.

The output of the Planner Manager is a command to each of its planners.

3.2. Planner

Planning is accomplished by means of a database of plan schemas. Encoded within the plan schema data structure is information on how to temporally decompose a task into subtasks, specific alternative actions to be investigated during the state space search, evaluation criteria to be used when determining the cost of alternative actions, and expected events which determine when to move on to the next action in the temporal sequence. Plan schemas also have synchronizing events for coordinating the plans generated by different planners. They are stored internally at each level of the planner hierarchy and do not flow between levels.

The basic architecture and search algorithms for each of the individual planners at a level is the same. What differs is the content of their output plan.

The Planner utilizes the A* search algorithm [3] to achieve an optimal path through its search space. Upon receipt of an input command, the planner retrieves the appropriate plan schema and then queries it for a list of possible next actions. The planner then hypothesizes these actions by posing them to the State Evaluator. Scores are assigned to each alternative by the State Evaluator by making queries to the world model regarding the effects of these commands on the vehicle. Characteristics such as traversability through a region of space, the amount of fuel required to perform the action, the time required to complete the action, and several others are applied to a predetermined cost function. The A* algorithm within the planner uses the resulting scores to incrementally build the least cost path to achieve its goal. The output of the planner is a plan graph that represents the temporal decomposition of its input command.

After the search goal has been achieved, the plan graph is passed to the Plan Update Module which substitutes this most recently generated plan for the one generated during the previous planning cycle. The new plan is more accurate than the last since this plan was generated with more recent world model data, thus making the plan generation process more effective. The Plan Update Module simply replaces unexecuted parts of the old plan with the newly generated one.

3.3. Executor

For each Planner there is an Executor that is responsible for successfully executing the plan prepared by its respective Planner. The Executor modules operate on short, regular intervals or execution cycles with the task of reacting to feedback. Plans generated by the Planner contain events signalling the completion of a task. The Executor monitors these specific events and informs the Planner Manager at the next lower level of the hierarchy when to move on to the next task. At the lowest level, the Executor operates as a servo control loop, sending out control signals to actuators and monitoring their response.

3.4. Plan Failure

When an unforeseen condition occurs during plan execution and the system can no longer operate within the conditions established by its present plan, emergency planning actions must occur. First, the executor initiates a reflexive action, such as halting the vehicle. This reflexive action attempts to avoid potentially hazardous situations. At this point, the executor signals plan failure to the planner. During its next cycle, the planner recognizes that the executor has failed and immediately switches to emergency replanning mode. When in an emergency recovery mode, many planning constraints, such as cycle times and resource limitations, are relaxed. The planner generates a plan to bypass the hazardous situation and continue towards the previous goal. All during emergency replanning, the world modeling and sensory processing systems are functioning normally, gathering data that allows the system to pick up where it left off.

Once the system has bypassed the hazardous situation, the planner switches back to normal planning mode and continues planning for the original command from its new state. Changes to plan graphs and data structures are transparent to the planner and sometimes occur across several levels of the hierarchy. However, even if radical changes to the global goals of the system were required to handle contingency situations, by the time control is passed back to the planner, it simply picks up the next task from the current input plan graph and continues cycling.

4. World Modeling

In this section, we focus on representing and maintaining the lake bottom map, with an emphasis on the concept of confidence-based mapping in an underwater environment from a sequence of data acquired by six sonar sensors (five forward-looking obstacle avoidance sonars and one downward-looking depth sonar). As the vehicle moves around in the underwater world, the information gained from the sonars is used to build an understanding of the environment. Each sonar reading is modelled as a cone, and the positions of the sonar sensors are assumed to be known.

The world model has two types of data 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 vehicle's current location. The global maps include terrain elevation data and several overlay features maps which include data on soil, vegetation, ridges, ravines, landmarks, obstacles, defense points, and transponders. The local maps include terrain elevation and its statistical features.

The region quadtree [4] is used to represent terrain elevation in the global maps. The advantages of using a quadtree are that large uniform areas in the map can be described compactly by a small number of large quadrants and that information retrieval is fast since the number of levels in the quadtree is related logarithmically to the resolution of the tree. In addition to the quadtrees used to represent elevation, 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 planning. Because the local maps are updated every time new sensor data are obtained, we represent them them as grid structures, which can be very efficiently updated.

4.1. Global Maps

The environment for the MAUV project is Lake Winnipesaukee in New Hampshire. A priori data from a survey of the lake bottom were collected and converted to quadtree format.

A separate sensor quadtree is used to store higher resolution depth values collected from the sonar sensors during vehicle runs. Both downward- and forward-looking sonars are used in refining the sensor map. A third quadtree stores a depth confidence value for each node in the tree. The confidence map supports the function of distinguishing spurious sonar readings caused by debris or signal inconsistencies from actual obstacles that must be detected and avoided. 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.

Point and line storing quadtrees [4] provide locations of known objects and topographic features of the lake bottom. These simplify 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.

4.2. Local Maps

Different levels of the control hierarchy require different local map resolutions. Also different types of data may be needed at each level. Generally, the resolution of the map at each level is about an order of magnitude less than the level below. All local maps are implemented as array data structures and only the lowest level (highest resolution) local map updates the global quadtrees. Figure 2 shows the mapping hierarchy for a generalized data set. 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. Because the updating algorithm only stores data in the sensor quadtree if the the confidence measure is above the level assigned to the a priori data, any node for which there are sensor data uses the sensed value. The local map uses a priori knowledge only if insufficient sensor data have been collected for that node. Confidence quadtree values are also copied into the local map.

In the current implementation, the mission level map divides the area into a coarse grid of approximately 25 x 25 pixels, each pixel storing the average depth of the corresponding area. 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, the e-move 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. For example, the output may be a percentage of pixels for which the vehicle 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 e-move pilot planner will query the world model for the traversability of several paths, using A* search to choose the best path. The e-move map is also the level updated directly by sensor readings; its modifications are propagated up through the mapping hierarchy.

4.3. The Updating Algorithms

At the beginning of a mission, the MAUV control system initializes the global and local maps, reading available a priori knowledge from a secondary storage device. 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 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 are collected.

In updating the map from downward-looking sonar data, the algorithm first computes an approximate neighborhood size of pixels to be updated around the current vehicle location which depends on the width of the sonar beam and the distance to the lake bottom. Given that the beam width 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 cause 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. For the depth sonar, all information is classifiable as either conflicting or agreeing with the knowledge already in the model. None of the data are irrelevant in this case.

The obstacle avoidance (forward-looking) sonar 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. 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 vehicle 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 3).

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 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 it 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.

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. Timing

An important issue for real-time control is timing of processes. In discussing the timing in the MAUV system, we consider the following factors at each level of the hierarchy: executor cycle period, input command update interval, replanning interval, and planning horizon.

The input command update interval is the rate at which new commands are input into a given level from the level above. The replanning interval is how often the planners at a given level do cyclic replanning.

The planning horizon is the amount of time into the future covered by a plan at a given level. The executor cycle period at each level is the rate at which the executor checks to see whether a new output command is to be sent to the level below. This cycle period is relatively fast. The following shows these values for each level of the hierarchy:

Mission Level	Replanning Interval	~30 min
	Planning Horizon	~2 hr
Group Level	Input Command Update Interval	~30 min
	Replanning Interval	~5 min
	Planning Horizon	~50 min
Vehicle Level	Input Command Update Interval	~5 min
	Replanning Interval	~1 min
	Planning Horizon	~10 min
E-move Level	Input Command Update Interval	~1 min
	Replanning Interval	~10 sec
	Planning Horizon	~2 min
Primitive Level	Input Command Update Interval	~10 sec
	Replanning Interval	~2 sec
	Planning Horizon	~20 sec
Servo Level	Input Command Update Interval	2 sec
	Replanning Interval	600 msec
	Planning Horizon	4 sec
	Output Command Update Interval	600 msec

The executor cycle period at each level is the same -- 600 msec. This is the rate at which new sensor data are collected. Therefore, the executor need not cycle faster than this since it will not determine that there can be a new output command unless new information about the world is known. The input command update interval increases by about a factor of five as we go up the hierarchy. The time values given in the table above represent approximate average times. For example, the rate at which new input commands can be received can be as fast as 600 msec (the executor cycle period) at any level. However, we do not expect this to happen very often.

The replanning interval at a given level is the same as the output command update interval at that level. In this way, the planners attempt to replan before each next command is determined.

The planning horizon at a given level is about twice the input command update interval at that level. Each planner therefore generates a plan that represents a decomposition of the current input command as well as the next input command.

6. Experimental Results

This section describes some intial experimental results on lake tests performed with one of the MAUV vehicles. The tests were performed at Lake Winnipesaukee, and were run using code at the servo, primitive, and e-move levels. The first experiment involved local obstacle avoidance. Figure 4 shows the path executed by the vehicle during a test run in which an obstacle was manually entered into the world model map at point C, and the vehicle was commanded to go from point A to point B. The control system succesfully planned and executed a path around the obstacle at point C.

The second experiment involved following along a predefined path. Figure 5 shows a raster-scan path from point A to point B. The vehicle determines its x,y position from acoustic navigation transponders which receive signals from navigation bouys placed in the water. The actual path executed by the vehicle during this run is shown in Figure 6. One of the obvious problems brought out by this run is that the vehicle tends to overshoot when it makes turns. This is a problem with the current low level control, which allows position control but not velocity control. Because the velocity is at maximum value when it takes a turn, it will always overshoot. Also, there is considerable error in the position measuring transponders, which largely accounts for the ragged appearance of the pathways.

The third experiment involved updating the internal model of the lake bottom with altitude information obtained from the downward looking depth sonar. Figure 7 shows three graphs. The top and middle graphs display the x and y positions, respectively, of the vehicle path. The bottom graph shows the lake depth values obtained from the world model along this path after the world model is updated from the information in the depth sonar.

7. Conclusion

This paper has described the planning and world modeling components of the MAUV architecture with a focus on requirements for navigation of the MAUV vehicles. Experimental lake test results have also been presented.

References

1. Albus, J. S. "A Control System Architecture for Intelligent Machine Systems." *IEEE Conf. on Systems, Man, and Cybernetics*, Arlington, VA, October 1987.

2. Herman, M. and Albus, J.S. "Overview of the Multiple Autonomous Underwater Vehicles (MAUV) Project." *Proc. 1988 IEEE International Conf. on Robotics and Automation*, Philadelphia, PA, April 1988, 618-620.

3. Nilsson, N.J. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, New York, 1971.

4. Samet, H. "The Quadtree and Related Hierarchical Data Structures." *ACM Computing Surveys*, June 1984, 187-260.

5. Simpson, J.A., Hocken, R.J., and Albus, J.S. "The Automated Manufacturing Research Facility of the National Bureau of Standards." *Journal of Manufacturing Systems*, Vol. 1, No. 1, 1983.

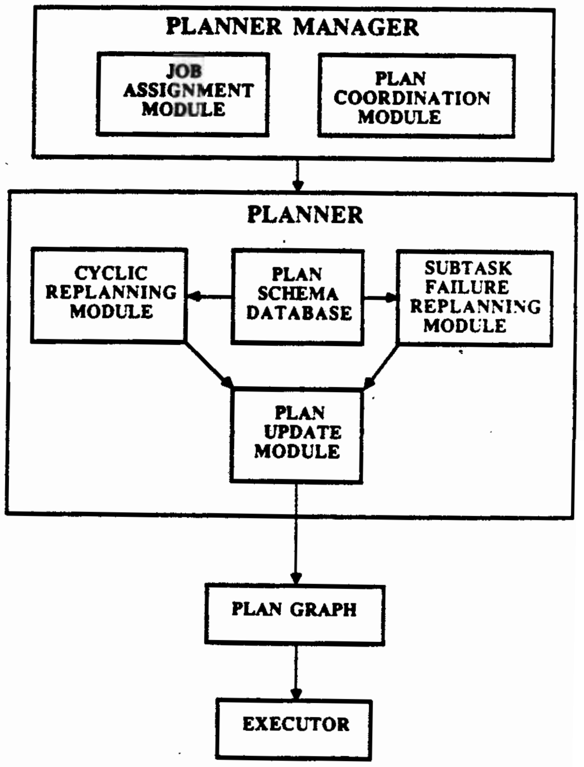

Figure 1. MAUV planning system.

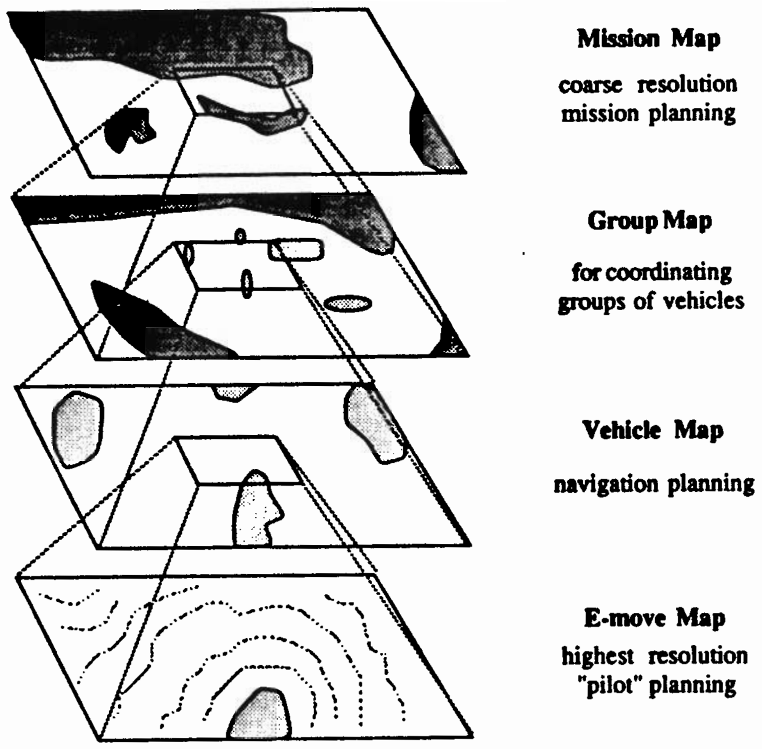
Mission Map
coarse resolution mission planning
Group Map
for coordinating groups of vehicles
Vehicle Map
navigation planning
E-move Map
highest resolution "pilot" planning

Figure 2. MAUV mapping hierarchy.

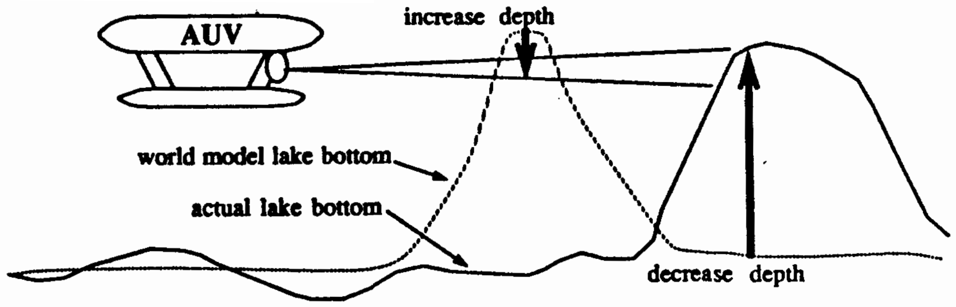

Figure 3. Updates from obstacle avoidance sensors remove false obstacles in the world model by increasing depth values in the map. Obstacles are added by decreasing the depth, in effect, raising the bottom of the lake model.

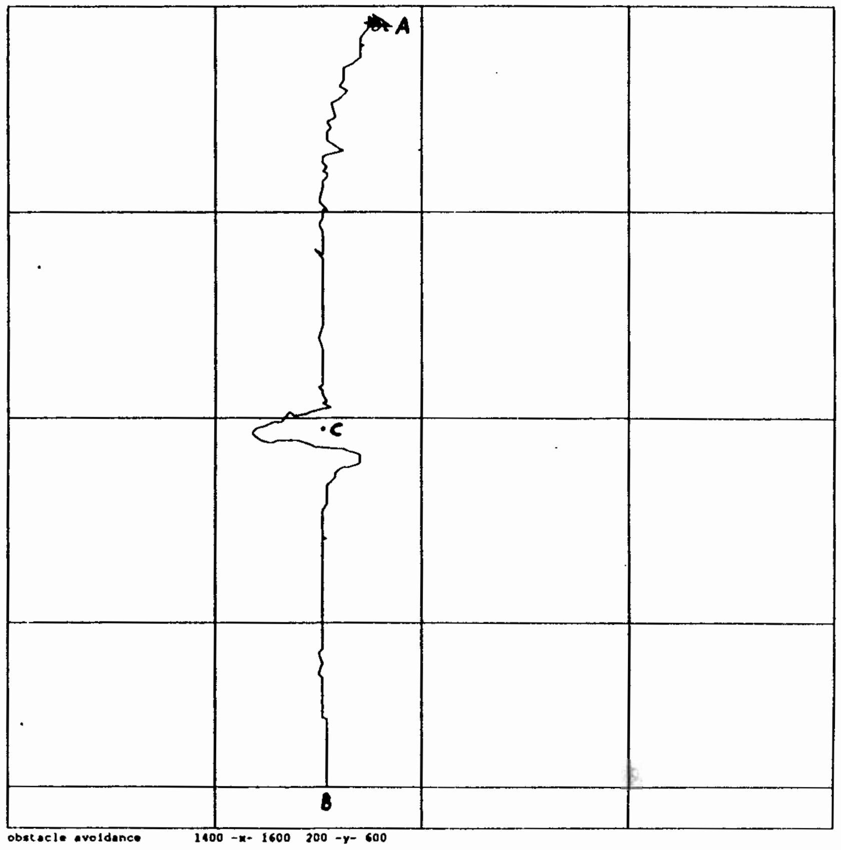

Figure 4. Obstacle avoidance test run.

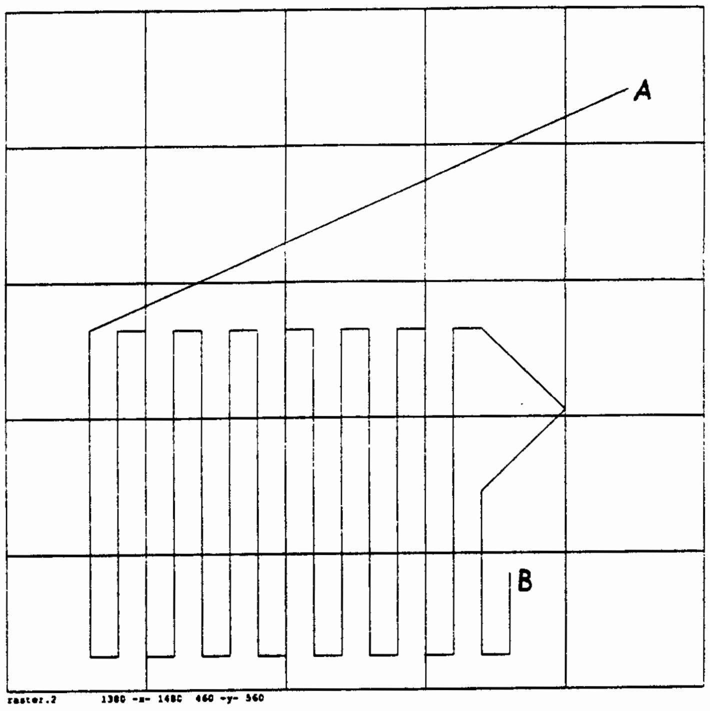

Figure 5. Predefined raster scan path.

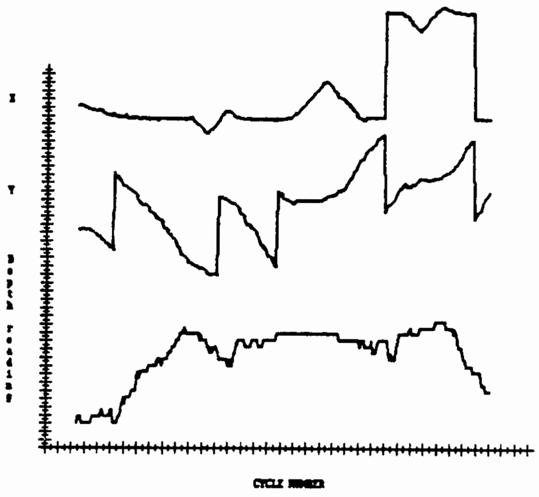

Figure 7. Updating the world model lake depth.

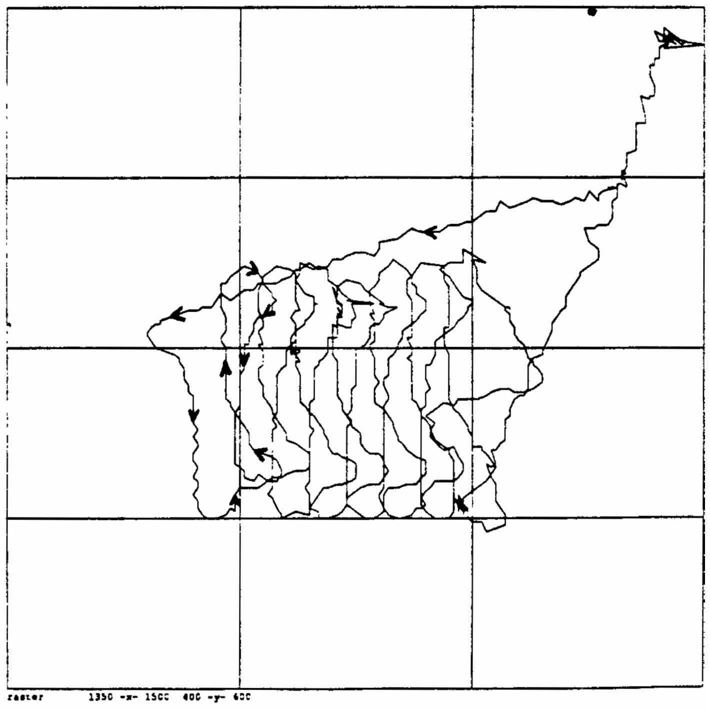

Figure 6. Actual raster scan path executed. Arrows show direction travelled.