

Autonomous Underwater Vehicle Path Planning: A Comparison of A* and Fast Marching Method

I. PROBLEM AND ITS IMPORTANCE

While most path planning algorithms such as A* and D* attempt to find an optimal path between two locations by minimizing a specified cost function, the standard path planning algorithms are ill-equipped to handle certain external environmental factors. This is especially true in underwater robotics where water current plays a significant role in the dynamics of the Autonomous Underwater Vehicle (AUV). Also, due to the fact that AUVs operate in low-bandwidth communication situations, knowledge of the effect of the water current on the vehicle's trajectory is vital to ensuring that the AUV reaches its goal safely. Finally, AUVs are typically nonholonomic in nature; thus, they are not completely controllable. The lack of complete control makes it difficult for the AUV to stop and turn as easily as its land-based robotic counterparts [1].

In order to alleviate the issues with underwater path planning and control, the Fast Marching Method (FMM) was developed which can be configured to factor water currents into path planning in order to minimize energy expended. Also, FMM acts as a "smoothing" path planning algorithm, which allows for more flexible control of the AUV [2].

II. RELATED WORK

As in other robotic fields, the A* approach to path planning has been explored for the underwater and three-dimensional paradigms by several authors [3], [4], [5]. Unfortunately, the A* method did not take water current into account. However, that changed when Sethian and Vladimirovsky wrote the seminal papers on Fast Marching Methods in application to underwater path planning [6]. The FMM can be used to solve a variety of anisotropic problems in optimal control, seismology, and paths on surfaces with an algorithmic complexity of $O(n \log(n))$ [6]. Several researchers used Sethian and Vladimirovsky's FMM to implement underwater path planning in simulation [1], [7], [8]. Hsun developed a comparative study between A* and FMM [9], which made the smoothing effect of FMM on paths obvious compared to A*. While Hsun mentioned the qualitative smoothness of the path generated by FMM, the author did not define a metric by which the smoothness could be measured and compared to the smoothness of the A* path.

III. APPROACH

The main purpose of the research was to measure the smoothness of the paths generated by several path planning algorithms. The path planning algorithms that were implemented were D*, A*, a modified variant of A*, and the Fast Marching Method (FMM). The second step was the development of a metric by which to measure the smoothness of the paths generated by each of the planning algorithms. Each of the

path planning algorithms as well as the method to measure the smoothness of the curves were implemented in GNU Octave, which is a MATLAB clone.

Before any path planning algorithms could actually be tested, a simulation of an environment with obstacles and free space had to be developed. Since Octave was the development tool, the obvious choice was to create a 2-dimensional 100-by-100 element matrix where each element in the matrix was a point in space. Each point in space could take on four different states: EMPTY, OBSTACLE, EXPLORED, and PATH. Elements categorized as EMPTY were valid locations for the path planning algorithms to utilize in order to complete the paths. Elements categorized as OBSTACLE were invalid locations for the path planning algorithms to utilize during path planning. The states labeled EXPLORED and PATH were assigned to elements after the algorithms completed. EXPLORED elements were elements that the path planning algorithms considered for a possible path, but did not use. The elements categorized as PATH were the elements that, when concatenated together, provided the final path solution for that specific algorithm. The complete procedure consisted of generating a map with EMPTY and OBSTACLE pixels, passing the generated map to the designated path planning algorithm, and displaying the image with the added EXPLORED and PATH pixels.

In order to test the basic functionality of the simulation environment, the first path planning algorithm that was implemented was the D* algorithm. This implementation of D* allowed for horizontal, vertical, and diagonal movement within a 2-dimensional grid where the cost of each move was the same, one unit. The D* algorithm worked by exploring all neighboring pixels, calculating the cost to arrive at each pixel, and then choosing the pixel with the lowest cost to explore next. This continued until the search arrived at the goal state. A priority queue was written in Octave in order to facilitate the "pushing" and "popping" of elements in to and out of the queue.

The next algorithm that was implemented was the A* algorithm. The A* implementation was basically the same as D*, except for the calculation of the cost function. In addition to calculating the current cost to reach a node, A* utilized heuristics to predict the minimal cost it would take to reach the goal from the current node. This additional cost helped the search algorithm to disqualify large groups of nodes that were not in the direction of the goal node. The first heuristic that was used was the straight line distance heuristic that merely measured the distance between the current node and the goal state. The heuristic and the current cost to reach the node are shown below.

$$h_{slid}(n) = \sqrt{x^2 + y^2} \quad (1)$$

$$g(n) = \text{Current} - \text{Cost} - \text{to} - \text{Node} \quad (2)$$

$$\text{Cost}(n) = g(n) + h_{std}(n) \quad (3)$$

A node that was closer to the goal state would have a smaller cost, thus, it would be higher in the priority queue than a node that was further away from the goal. During the experiment, the heuristic was modified to better predict the actual distance traveled in 2-dimensional grid-decomposed space by using the Manhattan Distance as well as diagonal distance. The new heuristic is displayed below.

$$h_1(n) = \max(\|cur.x - goal.x\|, \|cur.y - goal.y\|) \quad (4)$$

The heuristic for the A* algorithm was again modified to encourage the path planning algorithm to choose paths that were on the vector between the initial and goal states. This was accomplished by computing the cross product of the vector defined by the initial and goal states with the vector defined by the current pixel and the goal states. If the cross product result was large, this indicated that the current pixel was far from the line defined by the initial and goal points. Likewise, if the cross product was small, the current pixel was close to the line between the initial and goal states. The cross product was added to the previously defined heuristic, but not before being multiplied by the small value of 0.001 to ensure that the cross product did not dominate the heuristic. The additional heuristic calculation is provided below.

$$h_2(n) = h_1(n) + 0.001 * (\text{Vec}_{Cur-Goal} * \text{Vec}_{Init-Goal}) \quad (5)$$

The last path planning algorithm that was implemented was the Fast Marching Method. Similar to the D* algorithm, the FMM searched nearby neighbors, calculated the cost to reach each node from the initial state, and then chose the node with the smallest cost to expand upon. However, the FMM differs from D* and A* in the calculation of the cost to reach the node. While D* used a simple unit cost for the movement from one cell to the next, the FMM calculates the cost by approximating the solution to the Eikonal equation. In this 2-dimensional situation where the initial geometry of the agent is a single point, the calculation can be reduced to the following equation:

$$h_{FMM}(n) = \frac{1}{2} * \left\{ c_{n1} + c_{n2} + \sqrt{2 - (c_{n2} - c_{n1})^2} \right\} \quad (6)$$

where c_{nx} is the cost to reach neighbor node x . In the 2-dimensional implementation of the FMM, each new node has either one or two neighbor nodes that have already had their costs calculated. In the case where there is only one neighbor node, in order to calculate the cost of the new node, one is added to the cost of the neighbor node. In the case that there are two neighbor nodes, the cost is calculated using $h_{FMM}(n)$.

The last methods that were implemented dealt with calculating the smoothness of the paths that were created by the algorithms. The natural cubic spline was the first smoothness function that was explored. The natural cubic spline captures high frequency changes through the calculation of the second-derivative. Next it forces all derivatives to be greater than zero by squaring the second derivative. Finally, the resulting function is integrated to capture all of the squared values. In order

to implement this function as a discrete algorithm in Octave, the second-derivative was replaced by a $diff()$ function that calculated the incremental changes in the movement in the path from one step to the next. Next, the squaring function was replaced by the absolute value operator to ensure that positive and negative changes did not zero each other out. Finally, the integral was replaced by the cumulative summation operation. In Octave code, the final smoothness function operation was defined as: $cumsum(abs(diff(path)))$.

IV. EVALUATION

Each of the four algorithms were compared in four different scenarios. The scenarios consisted of varying the initial and goal states for each algorithm. The images for the first scenario are displayed in Figures 1, 2, 3, and 4 to demonstrate some of the characteristics of the algorithms. In the four images displayed, the path planning algorithms started at position (x=90 , y=40) and finished at position (x=5 , y=40). The OBSTACLE pixels are displayed in red, the EMPTY pixels are displayed in blue, the EXPLORED pixels are displayed in orange, and the final PATH pixels are displayed in bright green. Table I shows the smoothness metric for each path generated in the four different scenarios. The smoothness measurements for each algorithm were summed together in the fifth column.

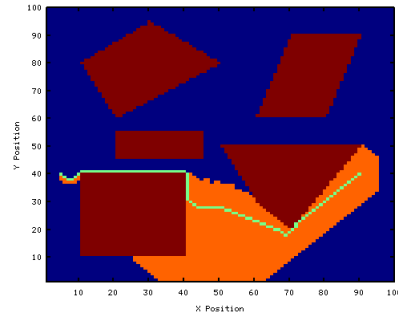


Figure 1. A* Path

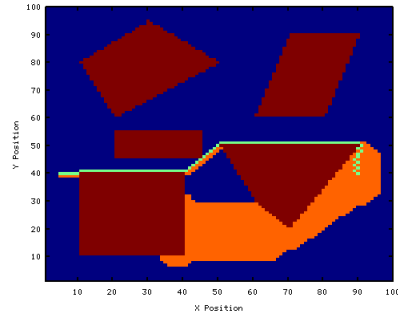


Figure 2. A* Modified Path

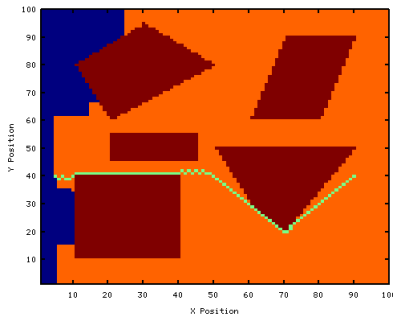


Figure 3. D* Path

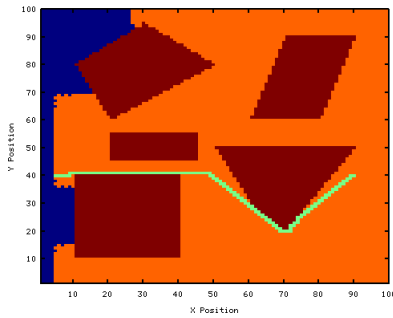


Figure 4. FMM Path

Table I
SMOOTHNESS MEASUREMENT OF EACH ALGORITHM

	1	2	3	4	Sum
A*	50	8	36	60	154
A* mod	22	0	20	60	102
D*	54	80	78	60	272
FMM	42	0	20	60	122

By observing the orange pixel areas it is obvious that the FMM searched more states than the A* and D* algorithms. Also, the A* modified algorithm generated a different path than the standard A*. According to Table I, all algorithms had the same smoothness metric for the fourth simulation and A* modified and FMM had the same smoothness metric for simulations two and three. The A* modified algorithm created the smoothest path with a total count of 102, while the D* algorithm proved to be the least smooth with a count of 272. The FMM algorithm was a close second with 122 counts.

V. DISCUSSION

Each of the four algorithms accomplished its main task of generating a path from the initial position to the goal position while avoiding obstacles. However, some of the paths were direct or smooth, while some of the paths made unnecessary vertical movements en route. While it might seem like these extra movements invalidated the algorithms, these extra movements were allowable due to the nature of the 2-dimensional grid space. On most simple paths there are actually several least-distance paths. This is most obvious in Figure 3 where the path oscillated up and down while moving diagonally across the map. These diagonal movements cost the same as

a horizontal movement; thus, steps are not wasted. This issue also occurred in the A* algorithm of Figure 1. However, the A* modified version reduced the number of oscillations by forcing the path to lie on the vector between the initial and goal states. As predicted, the FMM algorithm showed a great deal of smoothness without modifying the basic algorithm. One of the reasons why some of the column values in Table I were the same was because over long distances, there were fewer options for an optimal path, which allowed the number of vertical movements to converge.

Even though it was hypothesized that the FMM would generate the smoothest path, the modified A* algorithm was the smoothest. Also, A* searched fewer states than both FMM and D*. The A* modified algorithm was discovered during the implementation of the A* algorithm because the author was not satisfied with the initial paths generated by A*. The author learned a great deal about 2-dimensional grid space and how in simple situations, there are multiple optimal paths.

The D* and FMM algorithms both explored states radially; thus, they visited more states than the A* algorithm, which grew towards the goal all the time due to its heuristic function. In fact, it would be interesting to explore the use of heuristic functions with FMM to reduce the number of visited states.

The function to measure smoothness could be improved. The proposed path could be fitted to a polynomial curve and then the path could be measured for smoothness using traditional continuous function methods. Furthermore, the next iteration of the simulation could use water current data to fully test the FMM path planning.

REFERENCES

- [1] C. Petres, Y. Pailhas, Y. Petillot, and D. Lane, "Underwater path planning using fast marching algorithms," in *Oceans 2005 - Europe*, vol. 2, pp. 814–819 Vol. 2, 2005.
- [2] S. Garrido, L. Moreno, M. Abderrahim, and D. Blanco, "Robot navigation using tube skeletons and fast marching," in *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pp. 1–7, 2009.
- [3] K. P. Carroll, S. R. McClaran, E. L. Nelson, D. M. Barnett, D. K. Friesen, and G. N. William, "AUV path planning: An A* approach to path planning with consideration of variable vehicle speeds and multiple, overlapping, time-dependent exclusion zones," in *Autonomous Underwater Vehicle Technology, 1992. AUV '92., Proceedings of the 1992 Symposium on*, pp. 79–84, 1992.
- [4] C. Yao and Q. Guofeng, "Intelligent path planning in 3D scene," in *Computer Application and System Modeling (ICCSAM), 2010 International Conference on*, vol. 3, pp. V3–579–V3–583, 2010.
- [5] B. Garau, A. Alvarez, and G. Oliver, "Path planning of autonomous underwater vehicles in current fields with complex spatial variability: An A* approach," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 194–198, 2005.
- [6] J. Sethian and A. Vladimirov, "Ordered upwind methods for static hamilton-jacobi equations," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 98, no. 20, p. 11069, 2001.
- [7] C. Petres, Y. Pailhas, P. Patron, Y. Petillot, J. Evans, and D. Lane, "Path planning for autonomous underwater vehicles," *Robotics, IEEE Transactions on*, vol. 23, no. 2, pp. 331–341, 2007.
- [8] B. He, R. Hongge, K. Yang, L. Huang, and C. Ren, "Path planning and tracking for autonomous underwater vehicles," in *Information and Automation, 2009. ICIA '09. International Conference on*, pp. 728–733, 2009.
- [9] C. Chia Hsun, C. Po Jui, J. C. C. Fei, and L. Jin Sin, "A comparative study of implementing fast marching method and A* search for mobile robot path planning in grid environment: Effect of map resolution," in *Advanced Robotics and Its Social Impacts, 2007. ARSO 2007. IEEE Workshop on*, pp. 1–6, 2007.