Artificial Intelligence Journal , June 2004 (Under Revision)

Three-Dimensional Route Planner Using A* Algorithm; Application to Autonomous Underwater Vehicles

Nathan E. Brener *, Hua C. Looney, S. Sitharama Iyengar,
Qishi Wu, Narayanadas Vakamudi, Decai Yu, Qianyu Huang
Robotics Research Laboratory, Department of Computer Science
Louisiana State University, Baton Rouge, LA 70803 USA
Email: brener@bit.csc.lsu.edu, iyengar@bit.csc.lsu.edu

Jacob Barhen
CESAR Laboratory, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

ABSTRACT

We have developed a 3D route planner, called 3DPLAN, which employs the A* Algorithm to find optimum paths. The A* Algorithm has a major advantage compared to other search methods because it is guaranteed to give the optimum path. In spite of this significant advantage, no one has previously used A* in 3D searches as far as we are aware. The probable reason for this is that most researchers think that the computational cost of using A* for 3D route planning would be prohibitive. In this paper, we show that, on the contrary, it is quite feasible to use A* for 3D searches if one employs the new mobility and threat heuristics that we have developed. These new heuristics substantially speed up the A* Algorithm so that the run times are quite reasonable for the large grids that are typical of 3D searches.

We have adapted 3DPLAN to autonomous underwater vehicles (AUVs) by developing a map of realistic travel times that depend on ocean currents and the still water speed of the AUV. This version of 3DPLAN, called AUVPLAN, was used to calculate optimum paths in a region of the East China Sea. These optimum path calculations employed a travel time map with more than a million grid points and required less than 15 seconds of CPU time for each path. To our knowledge, this is the first time that actual travel times based on realistic ocean currents have been used in AUV route planning. We also used AUVPLAN to calculate optimum paths in the Persian Gulf, demonstrating that it successfully avoids underwater mines.

In order to compare the A* Algorithm to other route planning methods, we developed a Genetic Algorithm (GA) route planner and used it to perform optimum path calculations in the East China Sea. While the GA planner was able to find short optimum paths after several attempts, it failed to find any path at all for medium size and long paths. Thus the GA cannot handle the large maps that are typical of 3D route planning.

I. Introduction

Route planning has been widely used for civilian and military purposes. Three-dimensional (3D) route planning is especially useful for the navigation of autonomous underwater vehicles (AUVs) and combat helicopters. 3D route planning is a very challenging problem because the large grids that are typically required can cause a prohibitive computational burden if one does not use an efficient search algorithm. Several search algorithms have been proposed to perform 3D route planning, including case-based reasoning [4,7] and the genetic algorithm [5,6].

Case–based reasoning relies on specific instances of past experience to solve new problems. A new path is obtained by searching previous routes to see if there is one that matches the current situation in the features, goals and constraints. The new path is generated by modifying an old path in the previous path database using a set of repair rules. However, since the number of possible threat distributions is very large for most battle areas, it would not be feasible to store old routes for all or most of the possible threat arrangements. Thus the case-based method is not suitable for handling threats in route planning. In addition, when it has to synthesize complete new routes (in an area where no old paths are available) or modify old routes by synthesizing new segments, it doesn't use a guaranteed best-first search algorithm such as A*, but instead uses straight-line segments that go around obstacles. Thus the routes that it generates are neither locally nor globally optimal.

The genetic algorithm (GA) method is a stochastic search technique based on the principles of biological evolution, natural selection and genetic recombination. Genetic algorithms generate a population of solutions. Then such solutions mate and bear offspring solutions in the next generation. In this way, the solutions in the population improve over many generations until the best solution is obtained. However, genetic algorithms have been accepted slowly for research problems because crossing two feasible solutions does not, in many cases, result in a feasible solution as an offspring. Another disadvantage of GA is that although it can generate solutions to a route planning problem, it cannot guarantee that the solution is optimal (i.e., it can converge to a local, rather than a global, minimum).

The A* Algorithm [3] is a guaranteed best-first search algorithm that has been used previously in 2D route planning by several researchers including some of the authors of this paper [1,2]. A major advantage of the A* Algorithm compared to the other methods mentioned above is that A* is guaranteed to give the optimum path. In spite of this significant advantage, no one has previously used the A* Algorithm for 3D route planning as far as we are aware. The probable reason for this is that most people think that the computational cost of using A* for 3D route planning would be prohibitive. In this paper we show that, on the contrary, it is quite feasible to use A* for 3D searches if one employs the new mobility and threat heuristics that we have developed. These new heuristics substantially speed up the A* Algorithm so that the run times are quite reasonable for the large grids that are typical of 3D searches.

II. A* Algorithm and Our Implementation

1. Mobility and Threat Maps

Brener and Iyengar, in collaboration with Benton, have previously developed a two dimensional A* route planner [1,2], called the Predictive Intelligence Military Tactical Analysis System (PIMTAS), for military terrain vehicles such as tanks. Fig. 1 shows an example of a 2D mobility map (top) and 2D threat map (bottom) that were used as input to PIMTAS. The upper map in the figure is an actual mobility map of an area near Lauterbach, Germany, and the lower map is a prototype threat map which was generated in order to test the program. Both maps have 237 by 224 grid points. The mobility map has four types of GO regions represented by the colors green, light green, yellow-orange, and orange, which denote unlimited, limited, slow, and very slow areas, respectively. The mobility penalty for grid points in the unlimited, limited , slow, and very slow regions is 1, 2, 3, and 4, respectively. Thus the minimum mobility penalty at each grid point is 1. In general, the mobility map has three types of NO-GO regions represented by the colors red, blue, and white, which denote impassable obstacles, water, and urban areas, respectively. This last restriction follows military doctrine that urban areas are to be avoided. In the prototype threat map, each threat is modeled by a red inner circle where the vehicle is not allowed to go since it would almost certainly be destroyed if it came that close to the threat, and an orange outer circle where the vehicle is within range of the threat. The green regions are outside of the range of all of the threats. The threat penalty for each threat varies linearly from 1 to 0 as one goes radially outward from the boundary of the red circle to the boundary of the orange circle. If a grid point is within the orange circle of more than one threat, the threat penalty at that point is the sum of the threat penalties of all of the threats acting on that point. Grid points in the green regions have a threat penalty of 0.

In our new 3D route planner, which will be called 3DPLAN, the 2D mobility and threat maps described above have been extended to 3D in order to test the program. In 3DPLAN, the search region is represented by a digital map consisting of a Cartesian grid of points in which the step size in the x and y directions is the same but the step size in the z direction is in general different. Both the 3D mobility map and the 3D threat map have 237x224x150 grid points in the x, y, and z directions for a total of almost 8 million points. To our knowledge, this is the largest number of grid points that has ever been used in an A* search. In the 3D mobility map, each grid point in the GO regions has a mobility penalty of 1, 2, 3, or 4 depending on the mobility conditions. Grid points that are located in impassible areas are labeled as avoided points where the vehicle is not allowed to go. This test mobility map will be replaced with a realistic map of actual travel times when 3DPLAN is adapted to underwater vehicles. In our prototype 3D threat map, each threat is modeled by an inner sphere where the vehicle is not allowed to go and an outer sphere where the vehicle is within range of the threat. For each threat, the threat penalty varies linearly from 1 to 0 as one goes radially outward from the surface of the inner sphere to the surface of the outer sphere. As in the 2D case, if a grid point is within the range of more than one threat, its threat penalty is the sum of the penalties of all of the threats acting on that point. Grid points that are outside of the range of all of the threats have a threat penalty of 0.

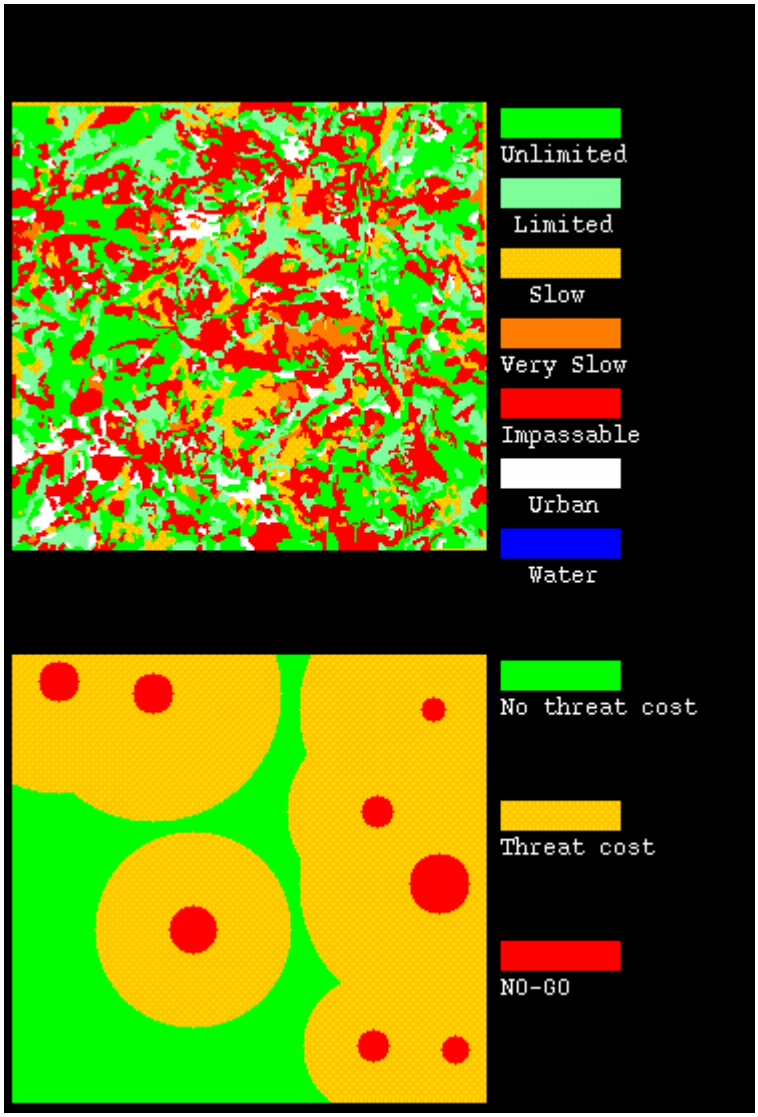

Figure 1. 2D mobility map (top) and 2D threat map (bottom)

Given the two maps, a starting point, and a target to be reached, the military planner enters a weight for each of the two path cost factors being considered: (1) mobility (travel time) and (2) threats. 3DPLAN will then quickly generate the lowest cost path from the starting point to the target, where the cost of the path is determined by multiplying the weight for each factor by the accumulated penalty for that factor. By entering a particular set of weights, the military planner can put any desired degree of emphasis on each of the cost factors. For example, a large weight for mobility and a small weight for threats would produce a fast path that may go close to enemy threats, while a large weight for threats and a small weight for mobility would produce a path that stays as far away from threats as possible and consequently may require a considerably longer travel time.

2. A* Algorithm

3DPLAN employs the A* Algorithm, in which the total cost, f, of the path that goes through a particular grid point on the digital map (this grid point will be referred to as the current point) is given by

$$f = g + h \quad (1)$$

where g is the actual cost that was accumulated in going from the starting point to the current point and h is an **underestimate** of the remaining cost required to go from the current point to the target. The heuristic h is the key quantity that determines how efficiently the algorithm works. h must not only be a guaranteed underestimate of the remaining cost, which ensures that no potential optimum paths will be discarded due to overestimating their total cost, but must also provide as close an estimate as possible of the remaining cost. The closer h is to the actual remaining cost, the faster the algorithm will find the optimum path. Thus the success of the algorithm depends on the choice of the heuristic h. With a proper choice of h, the algorithm can be highly efficient and can find the optimum path in a matter of seconds.

The actual accumulated cost, g, is given by

$$g = \alpha_m M + \alpha_t T \quad (2)$$

where

M = accumulated mobility penalty
T = accumulated threat penalty
α_m = mobility weight
α_t = threat weight

The weights α_m and α_t are entered by the military planner, as discussed above.

The accumulated mobility and threat penalties are given by

$$M = \Sigma R_i(M_{i-1} + M_i)/2 \quad (3)$$

$$T = \Sigma R_i(T_{i-1} + T_i)/2 \quad (4)$$

where the sum is over the grid points traversed in going from the starting point to the current point , M_i is the mobility penalty at grid point i, T_i is the threat penalty at grid point i, and R_i is the stepsize to go from grid point i-1 to grid point i.

The heuristic, which is an underestimate of the remaining cost required to go from the current point to the target, is given by:

$$h = \alpha_m h_m + \alpha_t h_t \quad (5)$$

where

h_m = underestimate of remaining mobility penalty (mobility heuristic)
h_t = underestimate of remaining threat penalty (threat heuristic)

We now describe the mobility heuristic and threat heuristic in detail.

3. Mobility Heuristic

Our mobility heuristic is different and better than the straight line heuristic used in previous A* approaches, since our new heuristic is larger than the straight line heuristic and still is an underestimate of the remaining cost. We will first describe our mobility heuristic in 2D and then extend it to 3D.

Fig. 2 shows a 2D mobility map that consists of a square grid of points. The step size in the x and y directions will be labeled r1 and the diagonal step size will be labeled r2, where r2=sqrt(2)*r1. In this figure, each grid point has a mobility penalty of either 1, 2, 3, or 4 (i.e, 1 is the minimum mobility penalty at each grid point) and P1, P2, and P3 are the start point, current point, and target point, respectively.

Let

n_x = number of steps in x direction between current point and target

n_y = number of steps in y direction between current point and target

Then the mobility heuristic h_m is given by:

$$h_m = \begin{cases} n_y r2 + (n_x - n_y) r1 & \text{for } n_x > n_y \\ n_x r2 + (n_y - n_x) r1 & \text{for } n_y >= n_x \end{cases} \quad (6)$$

In the example given in Fig. 2, the mobility heuristic to go from the current point P2 to the target P3 is

$h_m = 4*r2 + 2*r1$

This is larger than the straight line distance from P2 to P3, which is what other authors have used as the heuristic, and is still a guaranteed underestimate of the remaining mobility penalty to go from P2 to P3. Thus our mobility heuristic will cause the A* algorithm to run faster than it would with a straight line heuristic.

It is straightforward to extend this mobility heuristic to 3D. Fig. 3 shows a cell in the 3D map in which the step size in the x and y directions is the same but the step size in the z direction is in general different. The figure shows the 5 possible step sizes, labeled r1, r2, r3, r4 and r5, that the vehicle can take to go from a grid point to one of its neighbors, where

r1 = step size in x and y directions
r2 = sqrt(2)*r1
r3 = step size in z direction
r4 = sqrt(r1^2+r3^2)
r5 = sqrt(r2^2+r3^2)

Let

n_x = number of steps in x direction between current point and target

n_y = number of steps in y direction between current point and target

n_z = number of steps in z direction between current point and target .

The 3D mobility heuristic h_m is then given by

$$
\begin{aligned}
h_m = {} & n_x*r5 + (n_y-n_x)*r4 + (n_z-n_y)*r3 && \text{for } n_x <= n_y <= n_z \\
& n_x*r5 + (n_z-n_x)*r4 + (n_y-n_z)*r1 && \text{for } n_x <= n_z <= n_y \\
& n_y*r5 + (n_x-n_y)*r4 + (n_z-n_x)*r3 && \text{for } n_y <= n_x <= n_z \\
& n_y*r5 + (n_z-n_y)*r4 + (n_x-n_z)*r1 && \text{for } n_y <= n_z <= n_x \\
& n_z*r5 + (n_x-n_z)*r2 + (n_y-n_x)*r1 && \text{for } n_z <= n_x <= n_y \\
& n_z*r5 + (n_y-n_z)*r2 + (n_x-n_y)*r1 && \text{for } n_z <= n_y <= n_x
\end{aligned}
\qquad (7)
$$

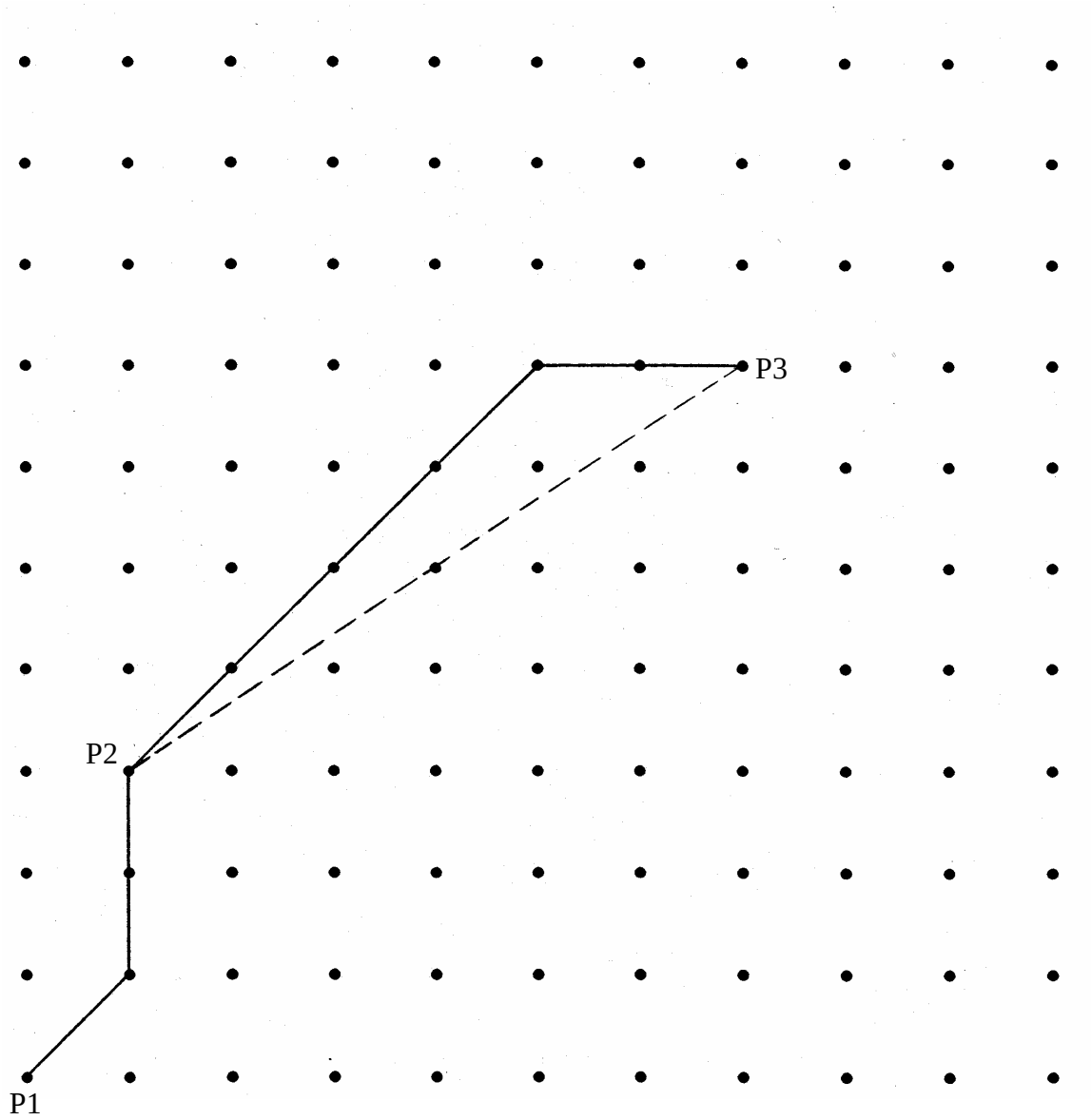

Figure 2. 2D mobility map consisting of a square grid of points

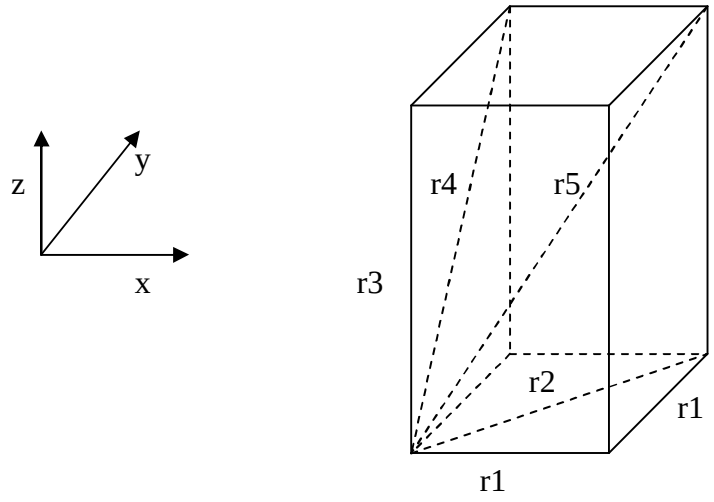

Figure 3. A cell in the 3D mobility map

4. Threat Heuristic

As far as we know, no other authors have ever used a threat heuristic in the A* Algorithm. The probable reason for this is that the minimum threat penalty is 0 rather than 1 (the minimum mobility penalty is 1 in our test map). Thus if one tried to use the same technique for the threat heuristic as was used for the mobility heuristic, the threat heuristic would be 0. In this paper we present a threat heuristic that is different from the mobility heuristic and is, in general, nonzero. Thus our new threat heuristic will speed up the A* Algorithm compared to not having a threat heuristic at all.

We will first describe our threat heuristic in 2D and then extend it to 3D. Fig. 4 shows the same square grid of points that was shown in Fig. 2, where P1, P2, and P3 are the start point, current point and target point, respectively. The threat heuristic h_t will be an underestimate of the remaining threat penalty to go from the current point to the target point. In order to construct this threat heuristic, we draw concentric squares around the target; these squares are labeled 1, 2, 3…, as shown in the figure, and the target point is square zero. In order to go from the current point to the target, the vehicle must visit each square at least once (i.e., it must visit at least one point in each square). In order to ensure that the threat heuristic is a guaranteed underestimate, we will use the minimum threat penalty in the square as the threat penalty of the point that the vehicle visits. The threat heuristic is then given by

$$h_t = r1(T_c + T_{n,min})/2 + \Sigma\, r1(T_{i,min} + T_{i-1,min})/2 \qquad (8)$$

where T_c is the threat penalty of the current point, $T_{i,min}$ is the minimum threat penalty in square i, n is the number of squares between the current point and the target, the sum is over all squares between the current point and the target, and we have multiplied by the smaller of the two step sizes that the vehicle can take, r1, in order to ensure that the threat heuristic is an underestimate.

It is straightforward to extend this 2D threat heuristic to 3D. In 3D we will use concentric rectangular boxes around the target. The vehicle will then have to visit each box at least once to go from the current point to the target. For the large boxes, the minimum threat penalty in the box may often be 0, depending on the distribution of threats. However, for small boxes (the ones close to the target), the minimum threat penalty in the box is less likely to be 0, especially if there is a dense distribution of threats around the target. In these cases, the threat heuristic will significantly speed up the A* Algorithm.

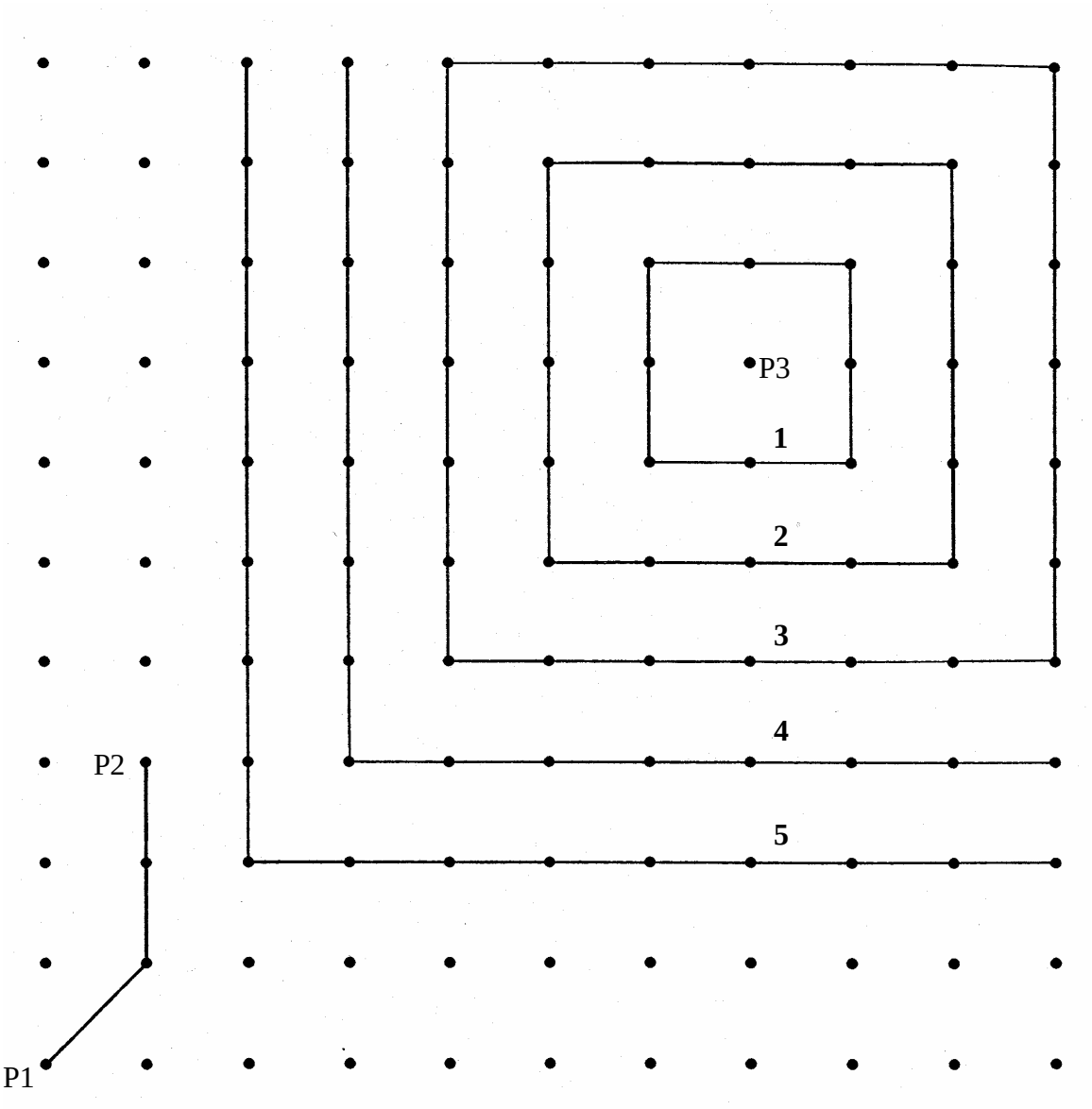

Figure 4. Threat heuristic in 2D map

5. Dynamic Threats

3DPLAN can also handle dynamic (changing) threats. If new threats appear or known threats disappear or move while the vehicle is traveling along its path, the military planner can quickly update the threat map. 3DPLAN will then rapidly generate a new optimum path from the current position to the target.

III. Test calculations and Comparison of Heuristics

In this section we give the results of some optimum path calculations using our new 3D A* route planner, 3DPLAN. In these sample calculations we used two different mobility maps, labeled M1 and M2, and two different threat maps, labeled T1 and T2. In the mobility map M1, all of the grid points in the GO areas have a mobility penalty of 1 (i.e., the mobility is uniform except for the obstacles), while in mobility map M2, the points in the GO areas have a mobility penalty of either 1, 2, 3, or 4. The threat maps T1 and T2 contain 24 threats and 26 threats, respectively. All four of these maps have 237x224x150 grid points for a total of almost 8 million points, which is the largest number of grid points that has ever been used in an A* search as far as we are aware. We used four different combinations of these maps: M1T1, M1T2, M2T1, and M2T2. For each of these combinations of a mobility map and a threat map, we calculated three different paths by choosing three different pairs of start/target points, which are given in Table 1. Thus altogether we calculated 12 different optimum paths in a search space of approximately 8 million grid points in order to test our new program. In all of the path calculations, the mobility weight α_m and threat weight α_t were both set equal to 1. For each of the 12 optimum paths, we did four different calculations using the following four combinations of the mobility heuristic and threat heuristic:

1) Our new mobility heuristic, our new threat heuristic
2) Our new mobility heuristic, no threat heuristic
3) Straight line mobility heuristic, our new threat heuristic
4) Straight line mobility heuristic, no threat heuristic

Combinations 1,2 and 3 enable us to compare our new mobility and threat heuristics with the heuristic that other authors have used in A* searches (combination 4).

These optimum path calculations were done on a Dell PC with a 3.06 GHZ Pentium IV processor. One of these optimum paths, Path 1 for the map combination T1M2, is shown in Fig. 5. The spheres in this figure are the inner spheres surrounding the threats, the rectangular solids are the obstacles in the mobility map, and the optimum path goes from the lower left to the upper right.

Tables 2-5 give the CPU time in seconds for the optimum path calculations described above. These tables show that when our new mobility and threat heuristics are used, all of the path calculations require less than one and a half minutes of CPU time. The tables also show that for the 12 test paths considered, our new heuristics reduce the CPU time by up to 67% compared to a straight line mobility heuristic and no threat heuristic, which is what other researchers have used in A* searches. In addition, the tables show that our new mobility heuristic alone reduces the CPU time by up to 60% and our new threat heuristic alone reduces the CPU time by up to 20%, compared to a straight line mobility heuristic and no threat heuristic, respectively. These

results demonstrate that our new mobility and threat heuristics significantly speed up the A* Algorithm.

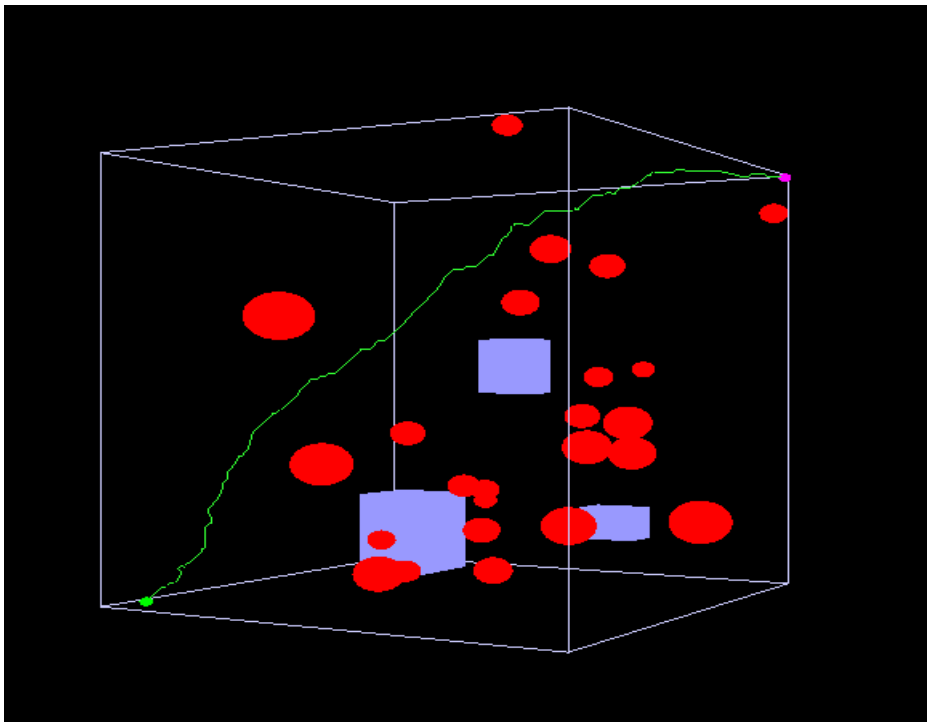

Figure 5. Path 1 for the map combination T1M2

Paths	Start Point	Target
Path 1	(6,22,0)	(236,218,149)
Path 2	(6,22,0)	(136,218,89)
Path 3	(100,0,50)	(236,218,149)

Table 1

Map T1M1

	Path 1	Path 2	Path 3
Our mobility heuristic, our threat heuristic	27.718	9.469	11.718
Our mobility heuristic, no threat heuristic	37.219	13.124	15.140
Straight line mobility heuristic, our threat heuristic	51.343	22.64	26.281
Straight line mobility heuristic, no threat heuristic	58.589	26.156	30.469

Table 2

Map T1M2

	Path 1	Path 2	Path 3
Our mobility heuristic, our threat heuristic	62.438	11.843	20.327
Our mobility heuristic, no threat heuristic	68.125	14.141	24.140
Straight line mobility heuristic, our threat heuristic	80.984	22.313	35.359
Straight line mobility heuristic, no threat heuristic	85.656	25.343	40.344

Table 3

Map T2M1

	Path 1	Path 2	Path 3
Our mobility heuristic, our threat heuristic	8.391	30.343	15.828
Our mobility heuristic, no threat heuristic	11.062	35.219	18.672
Straight line mobility heuristic, our threat heuristic	28.828	41.039	34.094
Straight line mobility heuristic, no threat heuristic	33.390	47.031	38.250

Table 4

Map T2M2

	Path 1	Path 2	Path 3
Our mobility heuristic, our threat heuristic	56.813	38.109	38.844
Our mobility heuristic, no threat heuristic	66.406	42.328	41.594
Straight line mobility heuristic, our threat heuristic	75.687	46.813	55.656
Straight line mobility heuristic, no threat heuristic	79.250	52.187	58.968

Table 5

IV. Application to Autonomous Underwater Vehicles

We now apply our 3D A* route planner to Autonomous Underwater Vehicles (AUVs). In this application, we replace the test mobility map described above with a map of realistic travel times which depend on ocean currents and the still water speed of the AUV. In the test calculations described in the previous section, the path cost depends on mobility penalties (to which the travel time is roughly proportional), but in the AUV application in this section, the path cost is equal to the actual travel time rather than just being approximately proportional to it. As far as we are aware, this is the first time that actual travel times based on realistic ocean currents have been used in AUV route planning.

In order to construct the map of travel times, we first created an ocean current map using the current velocity data from the Navy Coastal Ocean Model (NCOM) – East Asian Seas version, which was provided by the Naval Research Lab (NRL) at the Stennis Space Center in Mississippi. In this 3D ocean current map, there are 229×128×35 grid points in the x, y and z directions for a total of more than one million grid points. The x coordinate goes from longitude 115° E to 135° E in steps of 11.4 minutes of arc, the y coordinate goes from latitude 20° N to 30° N in steps of 11.4 minutes, and the z coordinate goes from 0 (the ocean surface) to a maximum depth of 4,655 m. This area includes a region of the East China Sea which contains Taiwan and neighboring islands.

At each grid point, the map gives the two components of the current velocity: the "U" velocity, which is in the x (East/West) direction, and the "V" velocity, which is in the y (North/South) direction, where East and North are positive and West and South are negative, and the velocities are in m/s. There is no current velocity in the z direction.

The 3D ocean current map described above is used to construct the 3D travel time map, which gives the times in seconds required for the AUV to go from each grid point to its 26 neighbors, taking into account the ocean currents and the still water speed of the AUV. Thus this map has 26 real numbers at each grid point. Fig 6 shows a grid point, labeled Y, and its 26 neighbors labeled A – X, Above, and Below.

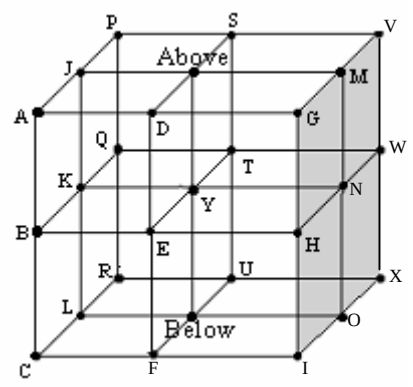

Figure 6. A grid point with 26 neighbors

The grid point Y will be called the central grid point and the points A – X will be referred to as follows:

A – southwest above, B – southwest same level, C – southwest below,
D – south above, E – south same level, F – south below,
G – southeast above, H – southeast same level, I – southeast below,
J – west above, K – west same level, L – west below,
M – east above, N – east same level, O – east below,
P – northwest above, Q – northwest same level, R – northwest below,
S – north above, T – north same level, U – north below,
V – northeast above, W – northeast same level, X – northeast below.

In this figure, the scale in the z direction is different from the scale in the x and y directions so that the grid can conveniently be displayed. In calculating the travel times to the N, S, E, W, NE, NW, SE, and SW neighbors in the planes above and below, we use only the component of the

distance in the xy plane, since the AUV can move up or down by simply changing its ballast and thus vertical motion that occurs at the same time as horizontal motion does not require any additional travel time. This assumption is valid as long as the vertical spacing between the grid points is substantially smaller than the horizontal spacing, which is the case here. Hence the difference in travel time to a neighbor in the plane above and the corresponding neighbor in the same plane (e.g., NE above and NE same level) is due only to the difference between the U and V velocities at the two neighbors and not to their distances from the central point.

In order to illustrate the calculation of the travel time map, we now give the formulas for the travel times from the central point to the three NE neighbors (NE same level, NE above, NE below), the three E neighbors (E same level, E above, E below), and the neighbors directly above and below.

NE neighbors

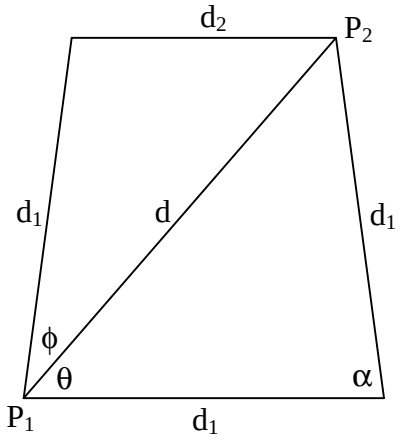

Figure 7. Calculation of the travel time from P_1 to P_2

In Fig. 7, P_1 is the central point, P_2 is the NE same level neighbor, d_1 is the grid point spacing for the latitude of P_1, and d_2 is the grid point spacing for the latitude of P_2. The grid point spacing for a given latitude gives the distance to neighboring points in the East, West, and North directions, while the distance to the South neighbor is given by the grid point spacing for the South neighbor's latitude, which is slightly larger. In other words, the spacing between the grid points slowly decreases as the latitude increases, which reflects the fact that the distance between latitude and longitude circles on the earth's surface gets smaller as the latitude gets larger. The angles α, θ, ϕ, and the distance d between the central point P_1 and the neighbor P_2 are given by

$$\alpha = \cos^{-1}\left(\frac{d_1 - d_2}{2d_1} \right) \tag{9}$$

$$\theta = \cos^{-1}\left(\sqrt{\frac{d_1 + d_2}{4d_1}} \right) \tag{10}$$

$$\phi = \alpha - \theta \tag{11}$$

$$d = \sqrt{(d_1)^2 + d_1 d_2} \quad (12)$$

The travel time t to go from P_1 to P_2 is then calculated as follows:

$$U_a = \frac{U_1 + U_2}{2} \quad (13)$$

$$V_a = \frac{V_1 + V_2}{2} \quad (14)$$

$$C_{12} = U_a\cos(\theta) + V_a\cos(\phi) \quad (15)$$

$$C_p = U_a\sin(\theta) - V_a\sin(\phi) \quad (16)$$

$$S_{12} = \sqrt{S^2 - (C_p)^2} \quad (17)$$

$$V_N = S_{12} + C_{12} \quad (18)$$

$$t = d / V_N \quad (19)$$

where

U_1 = U velocity at P_1
V_1 = V velocity at P_1
U_2 = U velocity at P_2
V_2 = V velocity at P_2
U_a = average of U velocities at P_1 and P_2
V_a = average of V velocities at P_1 and P_2
C_{12} = component of current velocity along the line joining P_1 and P_2
C_p = component of current velocity perpendicular to the line joining P_1 and P_2
S = AUV's still water horizontal velocity
S_{12} = component of AUV's still water horizontal velocity along the line joining P_1 and P_2
V_N = net velocity of AUV (the net velocity is along the line joining P_1 and P_2) .

The travel times to the NE above and NE below neighbors are calculated with the same formulas, where U_2 and V_2 are the U and V velocities at the NE above or NE below neighbor, and the distance d and angles α, θ and ϕ are the same as for the NE same level neighbor. As mentioned previously, we use only horizontal distances and velocities to calculate the travel times to neighbors in the above and below levels (except the ones directly above and below) because the travel time is determined by the horizontal motion. Vertical motion that occurs concurrently with horizontal motion does not add to the travel time. The travel times to the NW, SE and SW neighbors are calculated in a similar manner.

E neighbors

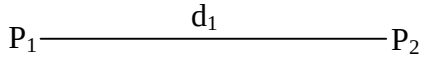

Figure 8. Calculation of the travel time from P_1 to P_2

In Fig. 8, P_1 is the central point, P_2 is the E same level neighbor, and d_1, the grid point spacing for the latitude of P_1, is the distance between P_1 and P_2. The travel time t to go from P_1 to P_2 is calculated as follows:

$$U_a = \frac{U_1 + U_2}{2} \tag{20}$$

$$V_a = \frac{V_1 + V_2}{2} \tag{21}$$

$$S_{12} = \sqrt{S^2 - (V_a)^2} \tag{22}$$

$$V_N = S_{12} + U_a \tag{23}$$

$$t = d_1 / V_N \tag{24}$$

These formulas are also used to calculate the travel times to the E above and E below neighbors. The travel times to the W, N and S neighbors are calculated in a similar manner.

Above and Below neighbors

The travel time to go from the central point to the neighbor directly above or below is given by

$$t = d / S_v \tag{25}$$

where d is the distance to the neighbor and S_v is the velocity of the AUV in the vertical direction (due to changing the ballast).

A* equations

When the travel time map, rather than the mobility map, is used, Equations 2, 3, and 5 become

$$g = \alpha_{tt} TT + \alpha_t T \tag{26}$$

$$TT = \Sigma TT_i \tag{27}$$

$$h = \alpha_{tt} h_{tt} + \alpha_t h_t \tag{28}$$

where

TT = accumulated travel time
α_{tt} = travel time weight
TT_i = travel time to go from grid point i –1 to grid point i .
h_{tt} = travel time heuristic (underestimate of remaining travel time needed to reach the target) .

The travel time heuristic h_{tt} is given by an expression similar to the one in Eq. (7):

$$
h_{tt} = \begin{cases}
n_x*t5 + (n_y-n_x)*t4 + (n_z-n_y)*t3 & \text{for } n_x <= n_y <= n_z \\
n_x*t5 + (n_z-n_x)*t4 + (n_y-n_z)*t1 & \text{for } n_x <= n_z <= n_y \\
n_y*t5 + (n_x-n_y)*t4 + (n_z-n_x)*t3 & \text{for } n_y <= n_x <= n_z \\
n_y*t5 + (n_z-n_y)*t4 + (n_x-n_z)*t1 & \text{for } n_y <= n_z <= n_x \\
n_z*t5 + (n_x-n_z)*t2 + (n_y-n_x)*t1 & \text{for } n_z <= n_x <= n_y \\
n_z*t5 + (n_y-n_z)*t2 + (n_x-n_y)*t1 & \text{for } n_z <= n_y <= n_x
\end{cases} \qquad (29)
$$

where

t1 = smallest travel time to go from a central point to an E, W, N, or S neighbor in the same plane
t2 = smallest travel time to go from a central point to a NE, NW, SE, or SW neighbor in the same plane
t3 = smallest travel time to go from a central point to a neighbor directly above or below
t4 = smallest travel time to go from a central point to an E, W, N, or S neighbor in the plane above or below
t5 = smallest travel time to go from a central point to a NE, NW, SE, or SW neighbor in the plane above or below .

These minimum travel times are obtained by scanning the entire travel time map.

Threats

Threats are handled the same way as described in Sec. II, where each threat is modeled by an inner sphere called the no-go sphere where the AUV is not allowed to go, and an outer sphere called the penalty sphere where the AUV is within range of the threat and hence incurs a threat penalty. Thus if the line between a central point and a neighbor passes through the no-go sphere of a threat, the AUV is not allowed to travel from the central point to that neighbor. If the threat is an underwater mine, it has a no-go sphere but no penalty sphere.

Sample Path Calculations

We used Microsoft Visual C++ 6.0 to develop 3DPLAN, the travel time map, and the AUV version of 3DPLAN, which will be called AUVPLAN. We then used AUVPLAN and the travel time map to perform optimum path calculations in a region of the East China Sea that contains Taiwan and neighboring islands. Figs 9-15 show the results of these optimum path calculations, which were done on a Dell PC with a 1.7 GHZ Pentium IV processor. All of the path calculations required less than 15 seconds of CPU time. Note that the optimum paths move up and down in order to find the most favorable currents. As discussed above, the use of the A* algorithm guarantees that the calculated paths have the shortest possible travel times. This cannot be guaranteed when other route planning algorithms, such as the genetic algorithm (GA), are used.

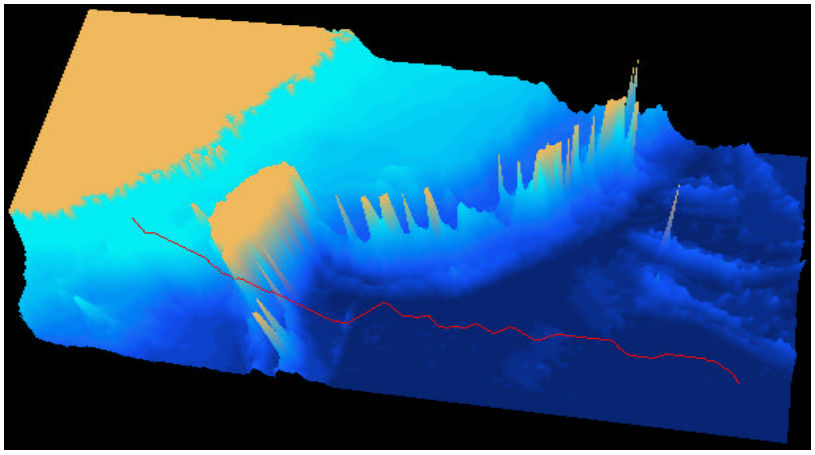

Figure 9: starting point (33,40,4), ending point (208,8,13)

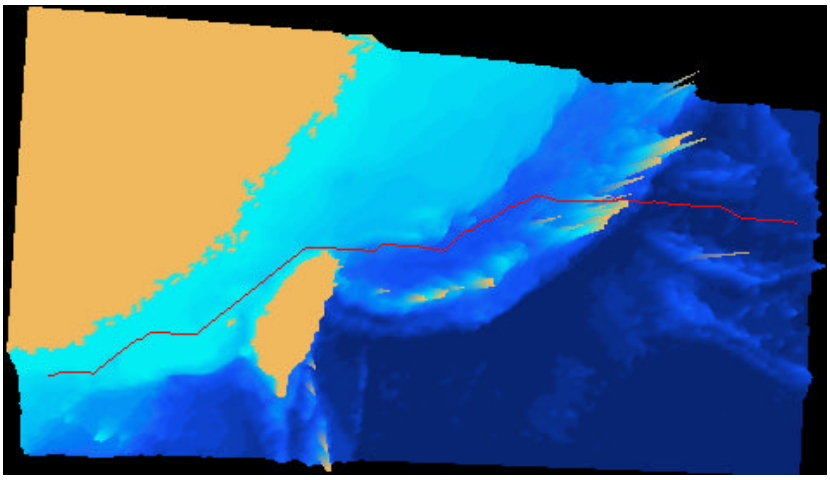

Figure 10: starting point (9,28,3), ending point (201,83,6)

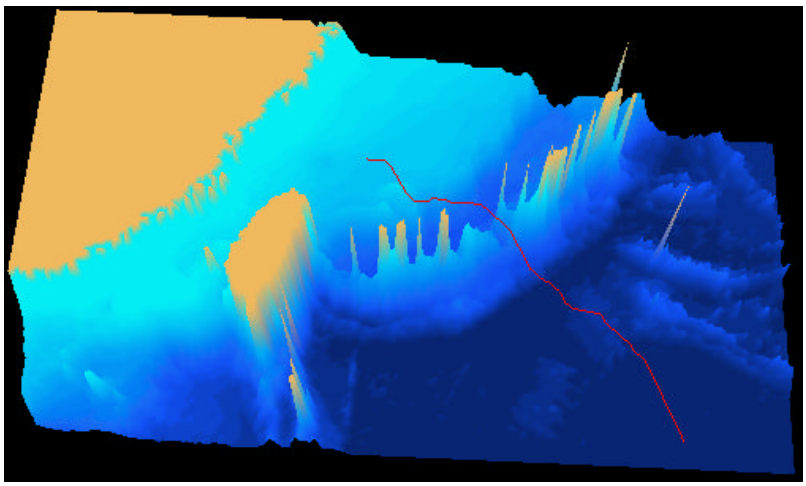

Figure 11: starting point (93,86,8), ending point (189,2,21)

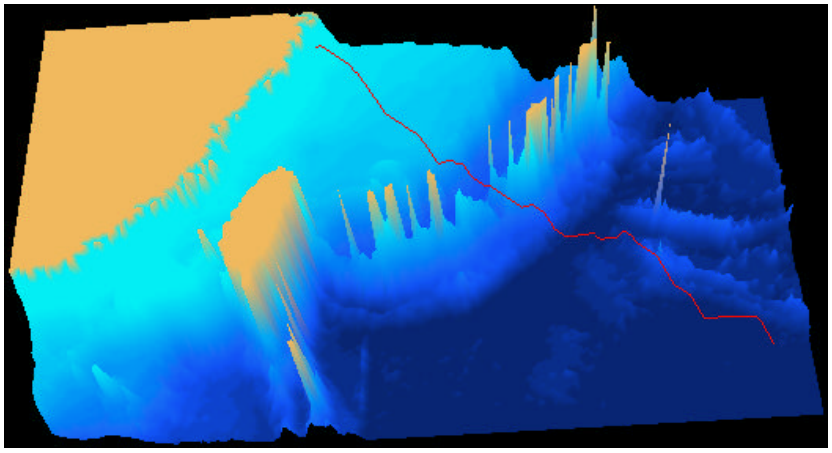

Figure 12: starting point (83,116,3), ending point (205,2,7)

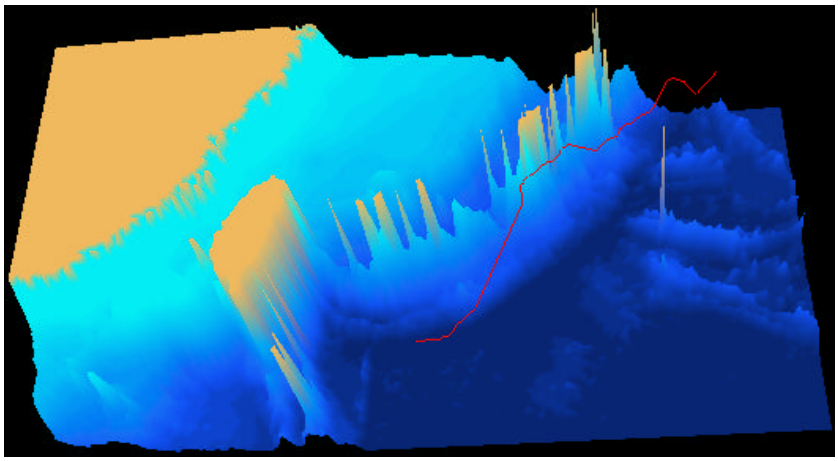

Figure 13: starting point (109,28,23), ending point (205,119,17)

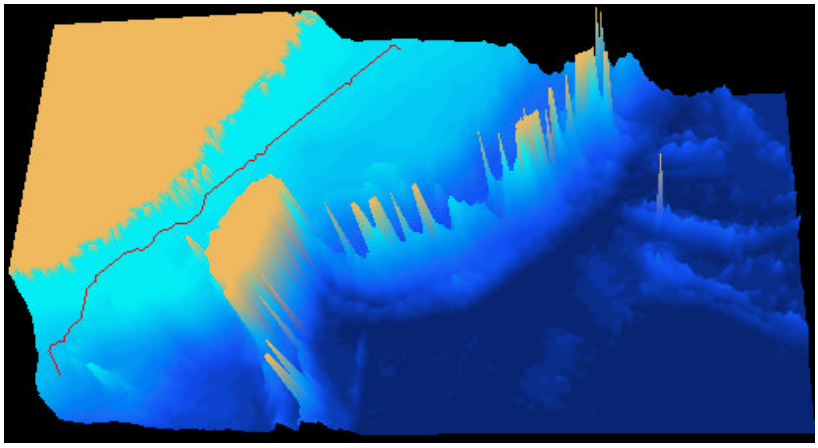

Figure 14: starting point (9,8,13), ending point (108,119,7)

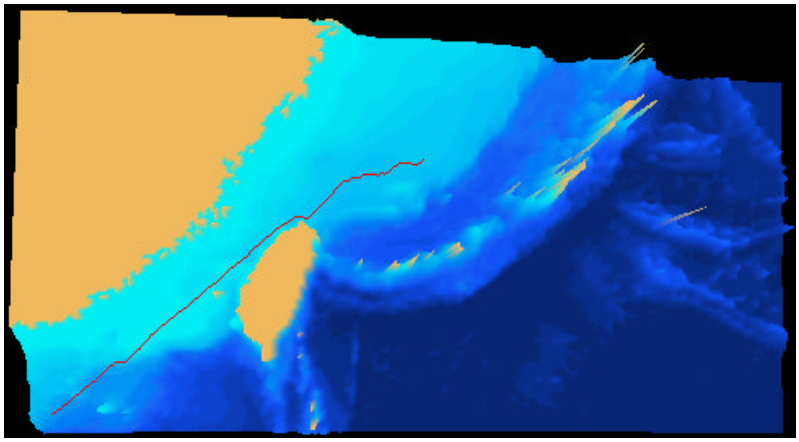

Figure 15: starting point (9,8,6), ending point (108,88,9)

Avoidance of Underwater Mines

If there are underwater mines in the region between the start point and the target, AUVPLAN will calculate the fastest path that avoids the no-go spheres around the mines. Also, if previously undetected mines are discovered near an optimum path that has already been determined, AUVPLAN will quickly calculate a new path that safely bypasses the mines. As an example, Fig. 16 shows an initial optimum path that was calculated in the Persian Gulf, previously unknown mines that lie close to this path, and the new path that maintains a safe distance from these mines. This new path is optimal subject to the constraint of avoiding the mines.

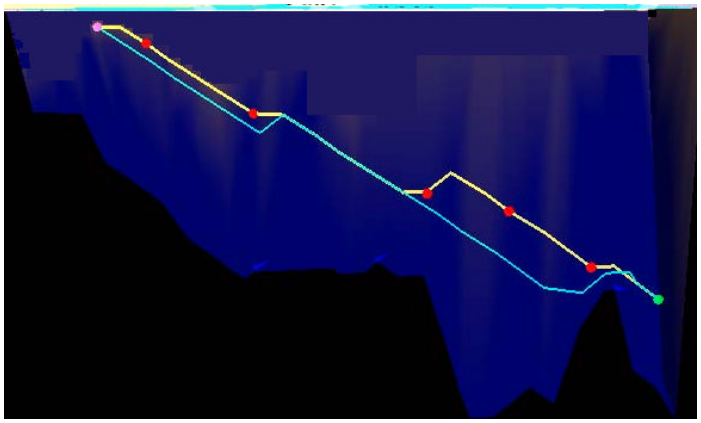

Figure 16: Two paths in Persian Gulf—one path goes close to the mines, while the new path avoids the mines.

V. Numerical Comparison of A* Algorithm and Genetic Algorithm

In order to compare the A* Algorithm to other methods, we developed a Genetic Algorithm (GA) 3D route planner and used both it and AUVPLAN to perform optimum path calculations in the East China Sea. In these calculations, we used a small map with 22×12×16 = 4,224 grid points and a medium size map with 110×60×25 = 165,000 grid points in addition to the full size map described above which has 229×128×35 = 1,025,920 grid points. For the full size map we considered 5 different paths, which are given in Table 6. For the small map and the medium size map, we used the first 3 paths and the first 4 paths, respectively. The results of these calculations are given below.

Paths	Start Point	Target
Path 1	(3,2,6)	(7,5,4)
Path 2	(4,6,7)	(15,9,11)
Path 3	(21,1,15)	(0,11,6)
Path 4	(109,59,19)	(0,1,5)
Path 5	(228,127,30)	(0,1,5)

Table 6

1. Small map (22´12´16 = 4,224 grid points)

Path 1: Start Point (3, 2, 6), Target (7, 5, 4)

	Path Cost (Travel Time)	CPU Time	Path Length
A* Algorithm	9.97 hours	0.015 seconds	5
Genetic Algorithm	9.97 hours	0.344 seconds	5

Path 2: Start Point (4, 6, 7), Target (15, 9, 11)

	Path Cost (Travel Time)	CPU Time	Path Length
A* Algorithm	20.42 hours	0.046 seconds	12
Genetic Algorithm	20.42 hours	19.546 seconds	12

Path 3: Start Point (21, 1, 15), Target (0, 11, 6)

	Path Cost (Travel Time)	CPU Time	Path Length
A* Algorithm	36.66 hours	0.062 seconds	22
Genetic Algorithm	Failed to find a path	N/A	N/A

2. Medium size map (110´60´25 = 165,000 grid points)

Path 1: Start Point (3, 2, 6), Target (7, 5, 4)

	Path Cost (Travel Time)	CPU Time	Path Length
A* Algorithm	9.97 hours	0.047 seconds	5
Genetic Algorithm	9.97 hours	0.610 seconds	5

Path 2: Start Point (4, 6, 7), Target (15, 9, 11)

	Path Cost (Travel Time)	CPU Time	Path Length
A* Algorithm	20.42 hours	0.079 seconds	12
Genetic Algorithm	20.42 hours	45.600 seconds	12

Path 3: Start Point (21, 1, 15), Target (0, 11, 6)

	Path Cost (Travel Time)	CPU Time	Path Length
A* Algorithm	36.66 hours	0.140 seconds	22
Genetic Algorithm	Failed to find a path	N/A	N/A

Path 4: Start Point (109, 59, 19), Target (0, 1, 5)

	Path Cost (Travel Time)	CPU Time	Path Length
A* Algorithm	197.46 hours	1.719 seconds	112
Genetic Algorithm	Failed to find a path	N/A	N/A

3. Full size map (229´128´35 = 1,025,920 grid points)

Path 1: Start Point (3, 2, 6), Target (7, 5, 4)

	Path Cost (Travel Time)	CPU Time	Path Length
A* Algorithm	9.97 hours	0.218 seconds	5
Genetic Algorithm	9.97 hours	0.579 seconds	5

Path 2: Start Point (4, 6, 7), Target (15, 9, 11)

	Path Cost (Travel Time)	CPU Time	Path Length
A* Algorithm	20.42 hours	0.219 seconds	12
Genetic Algorithm	20.42 hours	12.656 seconds	12

Path 3: Start Point (21, 1, 15), Target (0, 11, 6)

	Path Cost (Travel Time)	CPU Time	Path Length
A* Algorithm	36.66 hours	0.297 seconds	22
Genetic Algorithm	Failed to find a path	N/A	N/A

Path 4: Start Point (109, 59, 19), Target (0, 1, 5)

	Path Cost (Travel Time)	CPU Time	Path Length
A* Algorithm	197.46 hours	5.016 seconds	112
Genetic Algorithm	Failed to find a path	N/A	N/A

Path 5: Start Point (228, 127, 30), Target (0, 1, 5)

	Path Cost (Travel Time)	CPU Time	Path Length
A* Algorithm	383.55 hours	13.359 seconds	229
Genetic Algorithm	Failed to find a path	N/A	N/A

As shown above, GA was able to find the optimum path only for Paths 1 and 2, which are short paths with 5 and 12 nodes (grid points), respectively. In these calculations, the CPU time required for GA was considerably longer than for A*, particularly for Path 2 . Furthermore, as stated above, A* is guaranteed to find the optimum path in every calculation, but the same is not

true for GA. When GA finds a path, there is no way to know if the path is optimal unless A* has already been run for that start point and target. On the average, the GA planner had to be run twice to find the optimum path for Path 1 and 20 times to find the optimum path for Path 2. For Path 3, which is relatively short with 22 nodes, and Paths 4 and 5, which are longer with 112 and 229 nodes, the GA planner failed to find any path at all. Thus GA cannot handle the large maps that are typical of 3D route planning.

VI. Conclusion

We have developed a 3D A* route planner, called 3DPLAN, which runs efficiently for the large grids that are typical of 3D maps. The A* Algorithm has a major advantage compared to other search methods because it is guaranteed to give the optimum path. To our knowledge, this is the first time that A* has been used in 3D searches. The probable reason for this is that most researchers think that the computational cost of using A* for 3D route planning would be prohibitive. We have shown that, on the contrary, it is quite feasible to use A* for 3D searches as a result of the new mobility and threat heuristics that we have developed. These new heuristics substantially speed up the A* algorithm and make it a useful and efficient method for 3D route planning.

We have adapted 3DPLAN to autonomous underwater vehicles (AUVs) by replacing the test mobility maps that were used initially with a map of realistic travel times which depend on ocean currents and the still water speed of the AUV. This version of 3DPLAN, called AUVPLAN, was used to calculate optimum paths in a region of the East China Sea that contains Taiwan and neighboring islands. These optimum path calculations employed a travel time map with more than a million grid points and required less than 15 seconds of CPU time for each path. To our knowledge, this is the first time that actual travel times based on realistic ocean currents have been used in AUV route planning. AUVPLAN was also used to perform optimum path calculations in the Persian Gulf. These calculations demonstrate that AUVPLAN successfully avoids underwater mines.

In order to compare the A* Algorithm to other route planning methods, we used AUVPLAN and a Genetic Algorithm (GA) route planner to perform optimum path calculations in the East China Sea. While AUVPLAN quickly found the optimum path in every case, the GA planner was able to find the optimum path only for short paths and only after several attempts. For medium size and long paths, the GA planner failed to find any path at all. Thus the GA is not capable of handling the large maps that are typical of 3D route planning.

In the present version of AUVPLAN, the threats, such as underwater mines, are static and cannot attack the AUV from a distance. In the future, we plan to add the capability to avoid dynamic, long range threats such as enemy submarines and destroyers. We also plan to adapt 3DPLAN to manned underwater vehicles (submarines) and combat helicopters.

Acknowledgement

This work was supported in part by DOE-ORNL grant no. 4000008407 and by an NSF grant.

References

* To whom correspondence should be addressed.

[1] J. R. Benton, S. S. Iyengar, W. Deng, N. E. Brener, and V. S. Subrahmanian, "Tactical Route Planning: New Algorithms for Decomposing the Map", *International Journal on Artificial Intelligence Tools*, Vol. 5, Nos. 1 & 2, 1996, pp. 199-218.

[2] Nathan E. Brener, S. Sitharama Iyengar, and John R. Benton, "Predictive Intelligence Military Tactical Analysis System (PIMTAS)", Louisiana State University and U. S. Army Topographic Engineering Center.

[3] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", *IEEE Transactions of Systems Science and Cybernetics*, 4, July 1968, pp. 100-107.

[4] M. Kruusmaa and B. Svensson, "Combined Map-Based and Case-Based Path Planning for Mobile Robot Navigation", *Proceedings of International Symposium on Intelligent Robotic Systems*, January 10-12, 1998.

[5] John Smith and Kazuo Sugihara, "GA Toolkit on the Web", *Proceedings of 1st Online Workshop on Soft Computing*, August 1996, pp. 93-98.

[6] K. Sugihara and J. Yuh, "GA-Based Motion Planning for Underwater Robotic Vehicles", *Proceedings of 10th International Symposium on Unmanned Untethered Submersible Technology*, Autonomous Undersea Systems Institute, Durham, NH, 1997, pp. 406-415.

[7] C. Vasudevan and K. Ganesan, "Cased-Based Path Planning for Autonomous Underwater Vehicles", *Autonomous Robots*, Vol. 3, No. 2/3, 1996, pp. 79-89.