The Path Planning Algorithms for a Mobile Robot based on the Occupancy Grid Map of the Environment – A Comparative Study

Mijo Čikeš, Marija Đakulović and Ivan Petrović
Faculty of Electrical Engineering and Computing
University of Zagreb, Croatia
E-mail: {mijo.cikes, marija.dakulovic, ivan.petrovic}@fer.hr

***Abstract*—This paper presents a comparative study of three algorithms that use occupancy grid map of environments to find a path for a mobile robot from the given start to the goal – the D*, Two Way D* (TWD*) and E*. All three algorithms have ability of dynamic replanning in case of changes in the environment. The D* algorithm produces a path consisted of line segments that have discrete transitions between cell edges – a multiple of $45°$. This path is hard to follow by a mobile robot. The TWD* and E* algorithms produce more natural paths with continuous headings of the path line segments. The criteria for comparison were the path characteristics, the time of execution and the number of iterations of the algorithms' main while loop. The algorithms were verified both by simulation and experimentally on a Pioneer 3DX mobile robot equipped with a laser range finder.**

***Index Terms*—path planning, graph search, mobile robots.**

I. Introduction

The task of a path planning algorithm is to compute optimal path to the given goal and to replan the path in case the previously planned path is blocked by obstacles. The optimal path can be calculated by applying a classical graph search algorithm [8], such as the A* [4], D* [12] or focused D* (FD*) [13] algorithms. The D* and FD* algorithms are often used in path planning of mobile robots because of their capabilities of fast replanning in changing environments. Two-dimensional (2D) occupancy grid maps are usually used to represent a continuous environment by an equally-spaced grid of discrete points [14].

The path obtained by a classical graph search algorithm based on uniform resolution grid is a zigzag line with sharp turns, angles of which are limited to increments of $45°$. A mobile robot can not follow such a path smoothly due to its kinematic and dynamic constraints. To overcome this limitation, improved methods have been developed that use interpolation to produce paths not constrained to a small set of headings. Gennery in [3] proposed the algorithm that produces a path composed of long straight line segments with continuous headings by using the iterative end point fit method. This algorithm is based on the Witkowski's algorithm [15], which searches the undirected graph with equal weights in forward and backward directions by the breadth first search (BFS) and determines all optimal paths. Another interpolation based algorithm is the Field D* algorithm [2]. This algorithm is based on the D* algorithm and uses linear interpolation to derive the path cost of points between grid intersections. It generally produces more natural low-cost paths through grids with a range of continuous headings. The similar algorithm to the Field D* algorithm is the E* algorithm [7]. The E* algorithm extends the standard D* algorithm by using interpolation based on the Level Set Method [11] to produce the approximation of the real Euclidean distance from every cell in the search space to the goal. The E* algorithm gives the path that is very close to the optimal solution. In our previous work we proposed the algorithm called two-way D* (TWD*) algorithm based on the Witkowski's algorithm, which calculates the shortest possible path in the geometrical space [1]. The proposed algorithm also performs well in changing environments although it is computationally more demanding than the D* algorithm.

The aim of this paper based on a comparative study of the D*, TWD* and E* algorithms is two-fold. First, the path characteristics will be examined since the global geometric path must be further transformed to a robot trajectory by taking into account kinematic and dynamic constraints of the robot. An important issue when producing the trajectory for the mobile robot is the number of points in which path changes directions, since in these points the robot must slow down or even stop and turn in place. The sum of all angles accumulated in the points of path direction changes indicates how many turns a robot will have while following the path. Second, the algorithm real-time performance is very important in changing environments. It means that the path must be replanned as the environment changes to prevent collisions with the newly detected obstacles. The important parameters are the time of execution and the number of iterations of the algorithms' main while loop.

The rest of the paper is organized as follows. The formulation of the occupancy grid map and graph search is given in section II. In section III the algorithms D*, TWD* and E* were shortly described. Simulation results are given in section IV and experimental results in section V. Finally, the conclusion is given in section VI.

II. Problem statement

Algorithms D*, TWD* and E* uses undirected graph as the space of search, which is created from the occupancy grid map of the environment.

A. *Occupancy grid maps*

An occupancy grid map is created by approximate cell decomposition of the environment [6], [14]. The whole environment is divided into squared cells of equal size e_{cell}, which are abstractly represented as the set of M elements $\mathcal{M} = \{1, \dots, M\}$ with corresponding Cartesian coordinates of cell centers $c_i \in \mathbb{R}^2$, $i \in \mathcal{M}$. Each cell contains occupancy information of the part of the environment that it covers. In this paper two types of occupancy grid maps are used in the implementation of the algorithms: binary occupancy grid maps and weighted occupancy grid maps.

A binary occupancy grid map contains only free and occupied cells. Binary occupancy function $o(i) \in \{1, \infty\}$, $i \in \mathcal{M}$ is used for representing the set of all obstacles in the environment noted as $\mathcal{O} = \{i \in \mathcal{M} \mid o(i) = \infty\}$ and free environment is represented by the set of free cells noted as $\mathcal{N} = \mathcal{M} \setminus \mathcal{O}$, see Fig. 1. A weighted occupancy grid map contains free cells

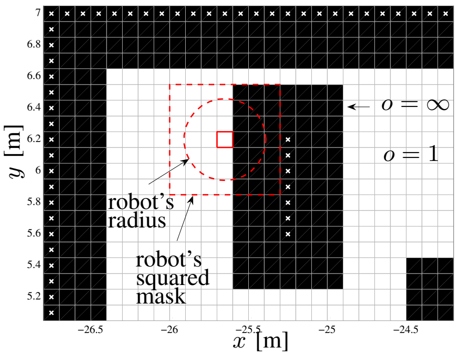

Fig. 1. A section of the experimental environment represented by the binary occupancy grid map. Free space is colored white ($o(\cdot) = 1$), and obstacles are colored black ($o(\cdot) = \infty$), where real obstacle position is marked by x. The robot's radius is $r_r = 0.26$ m, and the cell size is $e_{cell} = 0.1$ m.

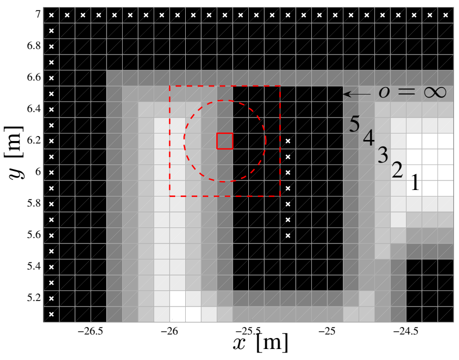

Fig. 2. A section of the experimental environment represented by the weighted occupancy grid map with the safety cost mask ($M_c = 4$). Free space is colored white ($o(\cdot) = 1$), unoccupied space within the safety cost mask is colored by shades of gray ($o(\cdot) \in \{2, 3, 4, 5\}$), and obstacles are colored black ($o(\cdot) = \infty$), where real obstacle position is marked by x.

($o(i) = 1$), occupied cells ($o(i) = \infty$) and also cells with other occupancy values ($1 < o(i) < \infty$). The cells with other occupancy values belong to the so-called safety cost mask which is introduced in the map around the obstacles to push the path away from the obstacles in order to assure safe robot motion. The size of the safety cost mask is defined by the integer number of cells M_c. The number of cells M_c influence on the distance between the path and the obstacles and can be determined as in our previous work [10]. Occupancy value of a cell within the safety cost mask depends on its distance from the closest occupied cell. The utmost cells of the safety cost mask obtain the occupancy value for one greater than unoccupied cells out of the safety cost mask ($o(i) = 2$), and occupancy values of inner cells incrementally increase from the utmost cells to the occupied cells. The safety cost mask defined in this way does not prevent the robot to pass through the narrow passages. The occupancy function of the occupancy grid maps with safety cost masks is defined as follows:

$$o(i) = \begin{cases} \max\{1, (M_c + 2 - \min_{j \in \mathcal{O}} \|c_i - c_j\|_\infty)\} & \text{if } i \notin \mathcal{O} \\ \infty \ \text{ if } i \in \mathcal{O} \end{cases} \tag{1}$$

where $\|\cdot\|_\infty$ is the infinity norm. Described procedure generates a smooth decrease of occupancy values from the obstacles towards the free space. Thus, the safety cost mask acts similarly as the artificial potential field [5]. Fig. 2 represents the weighed occupancy grid map with $M_c = 4$ cells wide safety cost mask of the same section of the environment as shown in Fig. 1.

B. *The robot representation in the grid map*

We assume that real shape of the mobile robot can be approximated by a circle of a radius r_r, which is very often used assumption in the literature. In that case the robot is represented by a squared mask in the grid map, within which the circular shape of the robot can be drawn. Thus, all obstacles in the grid map are enlarged for the integer number of cells $\lceil r_r/e_{cell} \rceil$, i.e. the robot is described by a squared mask of size $(2 \cdot \lceil r_r/e_{cell} \rceil + 1) \cdot e_{cell}$. Real shape of the robot and its squared mask in occupancy grid maps are depicted in Figs. 1 and 2. It can be noticed that at corners the real obstacles are enlarged $\sqrt{2}$ times more than strictly necessary, which additionally confirms the safety of trajectories that go through the free cells. In this way defined squared mask allows the robot to rotate in place at each point within the free space of the grid map and consequently the path planning algorithm needs to plan only the robot positions in the free space.

C. *Search graph*

Weighted undirected graph $\mathcal{G}(\mathcal{N}, \mathcal{E}, \mathcal{W})$ is created from the occupancy grid map in such a way that all unoccupied cells $\mathcal{N}$ represent the set of nodes in the graph. We say that two nodes $i, j \in \mathcal{N}$ in the graph are *neighbors* if $\|c_i - c_j\|_\infty = e_{cell}$. The set of edges is defined as $\mathcal{E} = \{\{i, j\} \mid i, j \in \mathcal{N}, i \text{ and } j \text{ are neighbors}\}$. The set of edge weights $\mathcal{W} = \{w_{i,j} \mid i, j \in \mathcal{N}, i \text{ and } j \text{ are neighbors}\}$ is defined as the cost of transition between neighbors as

$$w_{i,j} := \|c_i - c_j\| \cdot \max\{o(i), o(j)\}. \tag{2}$$

In binary occupancy grid maps there are two values of transitions between neighbors: straight transition (e.g. 10 cm) and diagonal transition (e.g. 14 cm). In weighted occupancy grid maps with safety cost mask transitions between neighbors are additionally weighted according to their distances to the obstacles.

III. Path planing

This section briefly restates the algorithms D*, TWD* and E*. More details can be found in [12], [7] and [1], respectively.

A. The D* algorithm

Stentz in [12] proposed a well known graph search algorithm capable of fast replanning in changing environments. It is also known as dynamic version of the A* algorithm without the heuristic function [4].

The execution of the D* algorithm can be divided into *initial planning* and *replanning* phases. Initial planning is performed if the robot is standstill at the start position and replanning is performed if the robot detects nodes with changed occupancy values during its motion.

For every searched node n, the D* algorithm computes the cost value $g(n)$ of the optimal path from the node n to the goal node and the value of the key function $k(n)$ for the replanning process, which stores the minimal value $g(n)$ before changes of weights in the graph $\mathcal{G}(\mathcal{N},\mathcal{E},\mathcal{W})$ happened. The algorithm stores the *backpointers* for every searched node n, which point to the parent node with the smallest cost g. A backpointer is noted by the function $b(\cdot)$, where $b(n) = m$ means that the node n has the smallest cost because it follows the node m. The backpointers ensure that optimal path from any searched node n to the goal node can be extracted according to the function $b(\cdot)$. The D* algorithm uses the so called *Open* list for storing the nodes that are examined at each algorithm iteration. The *Open* list is sorted by the key k. At each algorithm iteration the best node o in the *Open* list is removed from the list and to all its neighbors n the value $g(n)$ is calculated as

$$g(n) = g(o) + w_{n,o}, \tag{3}$$

and backpointer for the node n is set to $b(n) = o$, only if the previous value $g(n)$ is larger than the new value $g(o) + w_{n,o}$ (initially, all nodes have $g(n) = \infty$).

In the initial planning phase the D* algorithm starts the search from the goal node, examining neighbor nodes of minimal k value until the start node is reached. For better replanning performances, the exhaustive search can be done in the initial phase, which computes optimal paths and path costs g from every reachable node in the graph $\mathcal{G}(\mathcal{N},\mathcal{E},\mathcal{W})$ to the goal node. The nodes that are not reachable from the goal node (i.e. no path exists) have values $g(n) = \infty$. The node n has optimal value if $g(n) = k(n)$. At the end of initial planning phase all reachable nodes have optimal values g. The cost values g in the experimental environment are shown in Fig. 3 starting from the goal at $(155, 57)$. It can be noticed that the contours of equal values g are somewhere shaped as the octagon due to 8 neighbors and 8 different directions of traversing through the grid. The path obtained from the backpointers starting at $(7, 127)$ is shown in Fig. 4. The path is composed only of transitions through the grid in 8 directions.

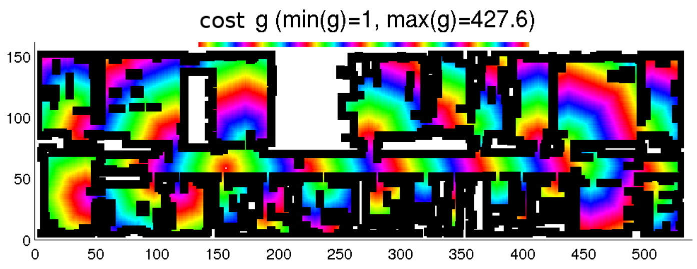

Fig. 3. Cost values $g(n)$ for exhaustive search of the graph by the D* algorithm.

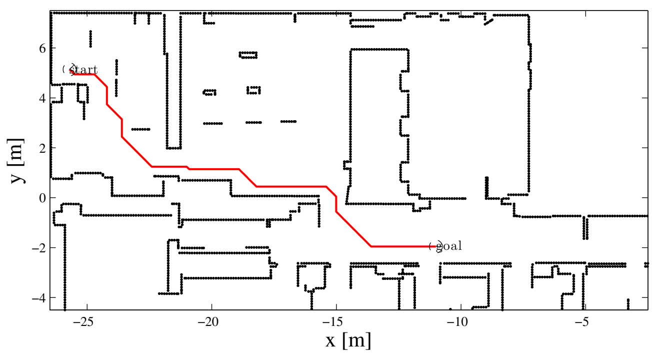

Fig. 4. The path calculated by the D* algorithm from the start $(7, 127)$ to the goal $(155, 57)$.

In the replanning phase due to some nodes change its occupancy values, first the weights in the graph $\mathcal{G}(\mathcal{N},\mathcal{E},\mathcal{W})$ are updated according to (2). All changed nodes are inserted in the *Open* list and those nodes propagate the change of the cost g to the parent nodes. The backpointers are redirected locally and the new optimal path from the robot's current position is determined. The number of expanded nodes is minimal and consequently the time of execution.

B. The TWD* algorithm

The two-way D* algorithm proposed in our previous work [1] is inspired by the Witkowski's algorithm. Like the Witkowski's algorithm it also searches the graph in forward and backward passes. The difference is that the TWD* algorithm uses the D* algorithm for graph search in these two passes instead of the breadth first search algorithm used in the Witkowski's algorithm. The usage of forward and backward passes of the D* algorithm through the graph enables the TWD* algorithm to find optimal paths also in weighted graphs, i.e. in the weighted occupancy grid maps with the cost mask.

TWD* like D* operates in two phases – initial and replanning phases. In the initial phase, the TWD* algorithm does the exhaustive search of the graph $\mathcal{G}(\mathcal{N},\mathcal{E},\mathcal{W})$ by applying one pass of the D* algorithm executed from the goal node to the start node (standard D*) and another pass executed from the start node to the goal node (the reverse D* (RD*)). For every node n in the graph $\mathcal{G}(\mathcal{N}, \mathcal{E}, \mathcal{W})$, D* calculates cost $g(n)$ to the goal node as well as $k(n)$ needed for path replanning and the backpointer $b(n)$ needed for reconstruction of the optimal path to the goal node. Analogously, RD* calculates cost $h(n)$ to the start node and $k_R(n)$ needed for path replanning and the backpointer $b_R(n)$ needed for reconstruction of the optimal path to the start node. At the end of both D* passes, costs g and h for every expanded node n give the complete cost of the path calculated as the sum, $f(n) = h(n) + g(n)$. Value $f(n)$ is the cost of the path from the start node to the goal node that passes through the node n and which is composed of two optimal paths: from the start node to the node n and from the node n to the goal node. The smallest value f is in all optimal paths from the start node to the goal node and is equal to $f_{min} = f(start) = f(goal)$.

The set of all nodes n such that $f(n) = f_{min}$ forms the geometrical area of minimal cost f_{min}. The path is calculated by partitioning the minimal cost area into smaller convex polygons and by finding the connection of polygon vertices that form the shortest path in geometrical space. The shortest path found by the TWD* algorithm consists of straight line segments with each straight line segment lying within the area of minimal cost f_{min}. The cost values f are shown in Fig. 5 calculated in the same experimental environment as shown in Fig. 3 with the goal at $(155, 57)$ and the start at $(7, 127)$. The area of equal values $f = f_{min}$ connects the start and the goal points. The contours of higher values of f are also shaped as octagon due to 8 neighbors and 8 different directions of traversing through the grid. The path calculated by the TWD* algorithm is shown in Fig. 6. The path is composed of straight line segments with arbitrary orientations.

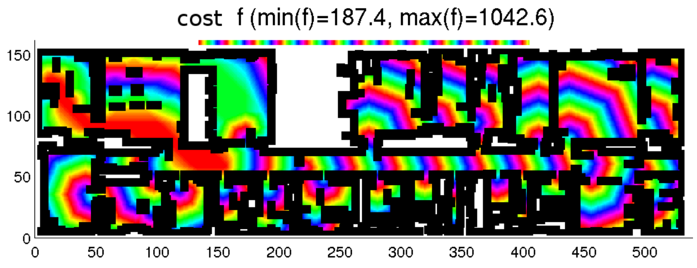

Fig. 5. Cost values $f(n)$ for the exhaustive search of the graph by the TWD* algorithm.

The replanning phase is initiated if nodes in vicinity of the moving robot change their occupancies. The TWD* algorithm replans the path by sequential execution of the D* and RD* searches of the changed graph $\mathcal{G}(\mathcal{N}, \mathcal{E}, \mathcal{W})$ and by the recalculation of the shortest path from the robot's current position (node R) to the goal node. Then, by calculating $f(n)$, the new minimal cost area is determined. In most cases the node R stays in the new minimal cost area, but, the change in the environment can be such that the node R is not in the new minimal cost area. That is due the fact that the value of f is

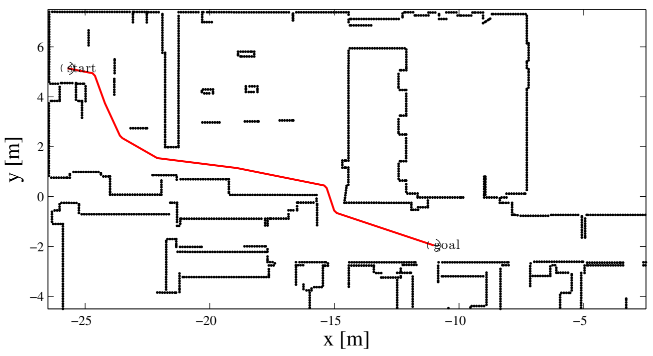

Fig. 6. The path calculated by the TWD* algorithm from the start $(7, 127)$ to the goal $(155, 57)$.

optimal according to the start node and the goal node, but not according to the robot's current position and the goal node. The new path is then composed of the part of the new D* path until the point which is both in the D* path and in the minimal cost area, and then, connected to the shortest part of the path to the goal node. This is the main drawback of the algorithm, since in some cases the replanned path is not the shortest possible path in the geometrical space, but is definitely shorter than the D* path.

C. The E* algorithm

The E* algorithm uses the interpolation function, which assigns numerical value to each grid map cell by the so called wavefront propagation over the free cells in the grid map [7]. The wavefront propagation acts like a continuous contour that sweeps from the goal node outwards and at each cell record the crossing time. Therefore, the crossing time at cells can be considered as samples of continuous navigation function. The path cost at each cell can be calculated by dividing the crossing time by the speed of the propagation.

The E* algorithm was described in grid map that stores information about the traversal risk at each cell. The risk function r is normalized to $[0, 1]$ where risk 0 corresponds to empty cells, and risk 1 corresponds to occupied cells. The lower r implies higher propagation speed. The occupancy function o given by (1) can be transformed to the risk function r by normalization

$$r(i) = \min\left(1, \frac{o(i) - 1}{M_c + 1}\right) \quad (4)$$

Therefore, for the example in Fig. 2 occupancy values $o(i) \in \{1, 2, 3, 4, 5, \infty\}$ are transformed to risk values $r(i) \in \{0, 0.2, 0, 4, 0.6, 0.8, 1\}$.

The E* algorithm calculates the crossing times based on the robust interpolation by the Level Set Method [11]. Unlike the D* algorithm, which uses one backpointer for a node n to represent which node is the next node in the optimal path $(b(n) = m)$, or on which neighbor node m the path cost $g(n)$ depends, the E* algorithm uses backpointers for one or two neighbors that are not obstacles and not lying on same axes. It has to be noted that the E* algorithm uses different definition for neighbors. Two nodes $i, j \in \mathcal{N}$ in the graph are *neighbors* if $\|c_i - c_j\| = e_{cell}$. In other words, each node has four neighbors in undirected graph, diagonal cells are omitted. Each node n has two values – $rhs(n)$ and $v(n)$. Initially, values of nodes are set to infinity. The one step looking ahead value is noted by $rhs(n)$ and the estimated value is noted by $v(n)$. Nodes that wait to be expanded are sorted in the queue called *Wavefront* by the key $\min(rhs(n), v(n))$. When the node n is subtracted from the queue the value $v(n)$ become equal to $rhs(n)$, and the node n is called locally consisted. Each node n has assigned the set $B(n)$, which contains nodes used in the computation of the node n and the set $D(n)$, which consists of nodes successors. The set $D(n)$ is used in the replanning phase, which is very similar to the D* algorithm. The interpolation function that calculates the crossing times of the node n is given by

$$(rhs(n), B(n)) = \text{ComputeValue}_{\text{LSM}}(n, Q(n)), \qquad (5)$$

where the set $Q(n)$ contains neighbor nodes of the node n that fulfills $\forall n \in Q(n) \mid v(n) < \infty$. The expression (5) calculates the backpointers as

$$B = \begin{cases} \{Q_1\} & \text{if } T_C - T_A \geq e_{cell}/F(n) \\ \{Q_1, Q_2\} & \text{otherwise} \end{cases} \qquad (6)$$

where the nodes $Q_1, Q_2 \in Q(n)$ are the best two neighbor nodes of the node n according to the value v. The values T_A and T_C are given by

$$T_A = v(Q_1),$$

$$T_C = \begin{cases} \infty & \text{if } Q = \{Q_1\} \\ v(Q_2) & \text{otherwise} \end{cases}$$

The expression (5) calculates the rhs value as $rhs(n) = T$, where the crossing time T is given by

$$T = \begin{cases} T_A + e_{cell}/F(n) & \Leftarrow T_C - T_A \geq e_{cell}/F(n) \\ \frac{1}{2}(-\beta + \sqrt{\beta^2 - 4\gamma}) & \text{otherwise} \end{cases} \qquad (7)$$

$$\text{where } \begin{cases} \beta = -(T_A + T_C) \\ \gamma = \frac{1}{2}(T_A^2 + T_C^2 - e_{cell}^2/F(n)^2) \end{cases}$$

The propagation speed F at the node n is calculated as $F(n) = 1 - r(n)$, with the risk function given by (4).

In case of binary occupancy grid map the estimated value $v(n)$ presents approximation of the shortest (Euclidean) distance from the node n to the goal node. The cost values v are shown in Fig. 7 calculated in the same experimental environment as shown in Figs. 3 and 5 with the goal at $(155, 57)$ and the start at $(7, 127)$. The contours of equal values of v are here shaped as circles since the cost estimates real shortest distance to the goal. The path calculated by the E* algorithm is shown in Fig. 8. The path is calculated as the gradient descent over the estimated cost v.

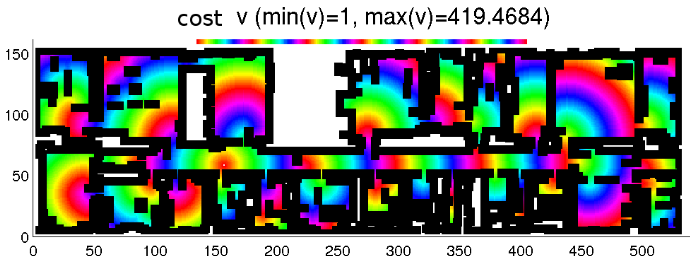

Fig. 7. Estimated values $v(n)$ for the exhaustive search of the graph by the E* algorithm.

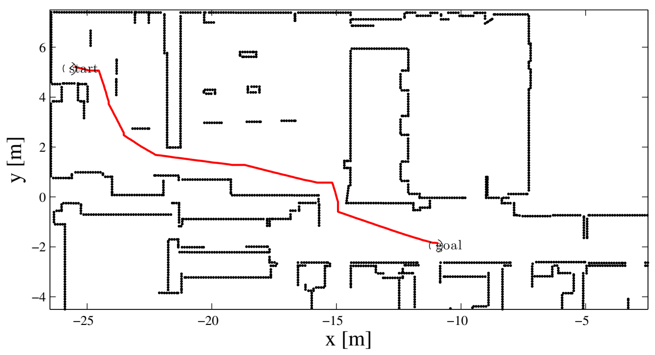

Fig. 8. The path calculated by the E* algorithm from the start $(7, 127)$ to the goal $(155, 57)$.

IV. Simulation Results

We tested the D*, TWD* and E* algorithms on three different binary occupancy grid maps. The first map is randomly generated environment shown in Fig. 9, the second map is the free environment shown in Fig. 10 and the third map is real experimental environment of our Department shown in Fig. 11. Simulation results are measured on notebook with Intel Core2Duo T7500 processor on Ubuntu 10.04 Linux operating system. Measurements were done in optimization mode (opt) with -03 flag on GNU C compiler. Path characteristics are l, $n\alpha_{init}$ and $\Sigma\alpha_{init}$ which denote lengths of the initial paths, the number of points in which the initial path changes direction, and the sum of all angles of the initial path direction changes, respectively. The execution parameters for the initial planning are the number of algorithm iterations noted by I and corresponding time noted by t_{init}. On the Department map we also tested dynamic planning in case of changes in the environment. On the same map we also tested behavior of algorithms when cell size changes. Finally, we tested algorithms in weighted occupancy grid map of the Department map.

A. Randomly generated map

The dimension of the randomly generated map is 500×500 cells, with the cell size of $e_{cell} = 0.1$ m and of about 50% occupied cells. Obstacles were created by drawing the squares at random places and of random sizes. Numerical results of the algorithms comparison is given in Table I, and the path comparison is shown in Fig. 9. Expectingly, the TWD* algorithm has the shortest path, but E* is very close to TWD*. E* gives approximation of the shortest (Euclidean) distance, as TWD* gives exactly Euclidean distance. The both E* and TWD* algorithm give significantly shorter path then the basic D* algorithm for approximately 6 %, but the time of execution t_{init} is worse for a factor 2.

TABLE I
COMPARISON OF ALGORITHMS IN THE RANDOMLY GENERATED MAP

Algorithm	I	t_{init} [ms]	$n\alpha_{init}$	$\Sigma\alpha_{init}$ [°]	l [m]
D*	217636	1331	23	1035	57.43
TWD*	435272	2424	7	69	53.41
E*	217636	2154	22	236	53.68

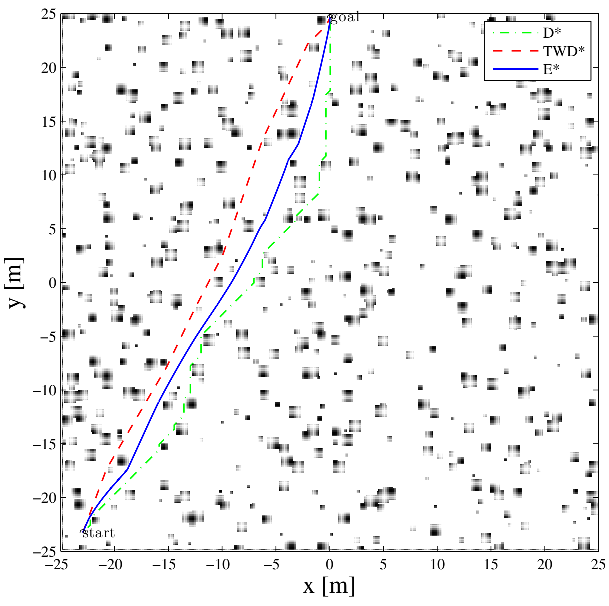

Fig. 9. Path comparison in the randomly generated map.

B. Free space

The dimension of the map is 165*540 cells, $e_{cell} = 0.1$ m. Numerical result are given in Table II and the path comparison is shown in Fig. 10. The results are similar to the ones obtained on the randomly generated map. The TWD* path is the shortest, and both E* and TWD* algorithm give significantly shorter path then the basic D* algorithm, but the time of execution t_{init} is worse for a factor 1.5. The minimal $n\alpha_{init}$

TABLE II
COMPARISON OF ALGORITHMS IN THE FREE SPACE

Algorithm	I	t_{init} [ms]	$n\alpha_{init}$	$\Sigma\alpha_{init}$ [°]	l [m]
D*	82150	338	17	675	28.46
TWD*	164300	484	11	100.2	26.513
E*	82150	486	26	290.01	26.526

and $\Sigma\alpha_{init}$ has TWD* algorithm. The E* has the largest $n\alpha_{init}$, but in $\Sigma\alpha_{init}$ is smaller than basic D* algorithm.

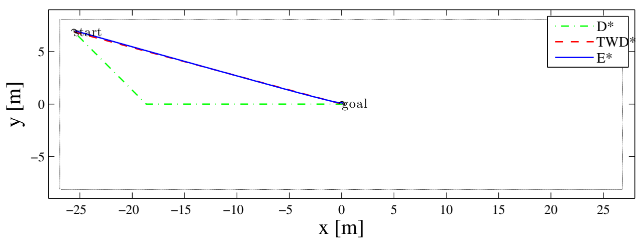

Fig. 10. Comparison of paths in the free space.

C. Department map

The dimension of map is 165*540 cells, $e_{cell} = 0.1$ m. Numerical results for initial planning are shown in Table III and the path comparison is shown in Fig. 11. The results are again similar to the ones obtained on the random and empty map. The TWD* algorithm produces the shortest path, the path is shorter than the basic D* algorithm for about 6 %. The E* algorithm has shorter path for about 4.5 % than the basic D* algorithm.

TABLE III
COMPARISON OF ALGORITHMS IN THE DEPARTMENT MAP

Algorithm	I	t_{init} [ms]	$n\alpha_{init}$	$\Sigma\alpha_{init}$ [°]	l [m]
D*	32724	26	15	675	18.81
TWD*	65448	102	11	239	17.65
E*	32724	119	22	530	17.95

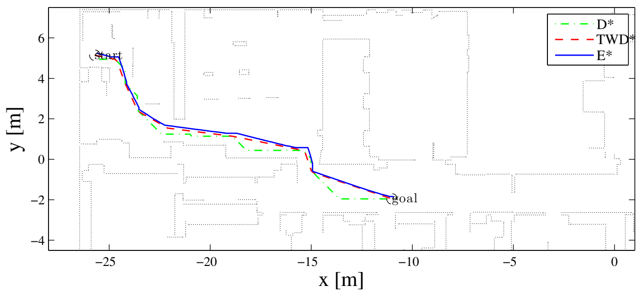

Fig. 11. Initial paths in the Department map.

Influence of the changing cell size are numerically shown in Table IV. When the cell size goes to zero, the path length is more appropriate and become very similar to the path given by the TWD* algorithm. The basic D* and the TWD* algorithms do not significantly change path length by changing the cell size. Paths given by the E* algorithm for different cell size are shown in Fig. 12.

Table V shows times of replanning in the Department map. t_{replan}^{max} denotes the highest time of replanning during the robot motion through the environment populated by unknown obstacles. The number of iterations of the highest replanning is denoted by I_{replan}^{max}. The sum of all replanning times is denoted by Σt_{replan} and ΣI_{replan} is the sum of all iterations during replannings. E* and TWD* have similar times of replanning.

TABLE IV
COMPARISON OF ALGORITHMS FOR THE CELL SIZE $e_{cell} = 0.05$ AND $e_{cell} = 0.025$.

Algorithm	I	t_{init} [ms]	l [m]
D*($e_{cell} = 0.05$ m)	138162	218	18.68
TWD*($e_{cell} = 0.05$ m)	276324	448	17.46
E*($e_{cell} = 0.05$ m)	138162	672	17.61
D*($e_{cell} = 0.025$ m)	596148	2732	18.53
TWD*($e_{cell} = 0.025$ m)	1192296	4501	17.30
E*($e_{cell} = 0.025$ m)	596148	5097	17.37

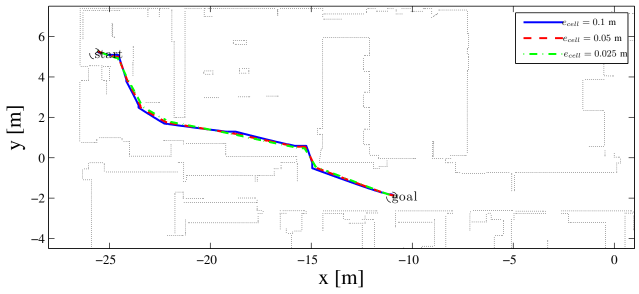

Fig. 12. Initial paths of the E* algorithm in the Department map for the different cell sizes.

E* has slightly better sum of all replanning times than TWD*. The basic D* algorithm has the best replanning time and t^{max}_{replan} is better approximately for a factor two. Fig. 13 shows path lengths at each step of replanning. The TWD* algorithm is not always optimal in the replanning process. The TWD* algorithm gives the shortest path until the step 55. At the step 55 the TWD* algorithm is not optimal and the E* algorithm has the shortest path. This case is illustrated in Fig. 14.

TABLE V
REPLANNING EXECUTION

Alg.	t^{max}_{replan} [ms]	I^{max}_{replan}	Σt_{replan} [ms]	ΣI_{replan}
D*	11	9567	18	18051
TWD*	20	18885	139	75344
E*	22	6676	115	30943

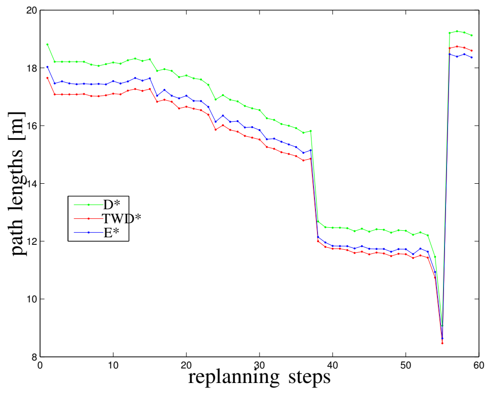

Fig. 13. Path lengths at each step of replanning.

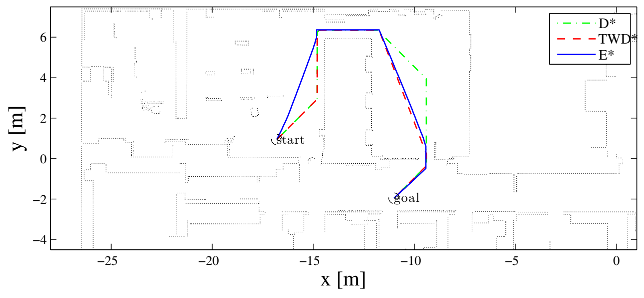

Fig. 14. The path lengths in the replanning process at the step 55.

Finally, the comparison of algorithms in the Department map with safety cost map is shown in Table VI. Here the E* algorithm produces the shortest path, but $n\alpha_{init}$ and $\Sigma\alpha_{init}$ are considerably higher than for TWD*. The path is smoother and slightly distanced from the obstacles for the safety cost mask. The paths are shown in Fig. 15.

TABLE VI
COMPARISON OF ALGORITHMS IN THE DEPARTMENT MAP WITH SAFETY COST MAP

Algorithm	I	t_{init} [ms]	$n\alpha_{init}$	$\Sigma\alpha_{init}$ [°]	l [m]
D*	32727	46	27	1215	19.88
TWD*	65456	141	22	593	19.12
E*	32727	150	61	1151	19.09

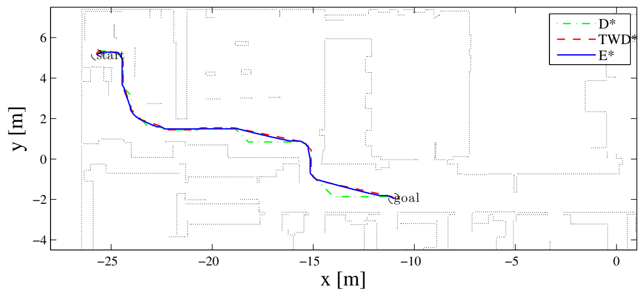

Fig. 15. Initial paths in the Department map with the safety cost mask.

V. EXPERIMENTAL RESULTS

The experimental results were obtained with a Pioneer 3DX mobile robot at our Department. The occupancy grid map with the safety cost mask were used. The laser range finder SICK LMS200 mounted on the robot was used for environment perception. The dynamic window based algorithm, described in our previous work [9], was used for path following.

Fig. 16 shows the initial and replanned paths calculated by the E* algorithm, and the robot's trajectory while following the path. The comparison of the path lengths of all three algorithms at each replanning step is shown in Fig. 17. It can be seen that the path calculated by the E* algorithm is the shortest in all replanning steps.

Numerical results of the initial planning is shown in Table VII, and of the dynamical planning is shown in Table VIII.

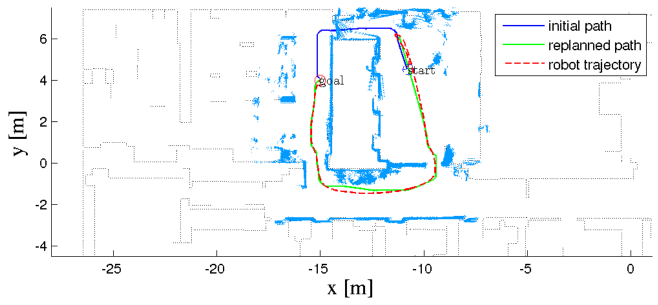

Fig. 16. The initial and replanned paths calculated by the E* algorithm and the trajectory driven by the mobile robot Pioneer 3DX.

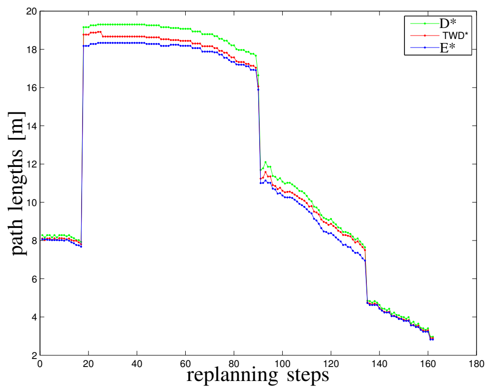

Fig. 17. Comparison of the path lengths at each replanning step.

It can be seen that the sum of all replannings is much higher comparing to the simulation results, which is due to incomplete map of the environment.

TABLE VII
COMPARISON OF ALGORITMHS IN DEPARTENT MAP WITH SAFETY COST MAP.

Algorithm	I	t_{init} [ms]	$n\alpha_{init}$	$\Sigma\alpha_{init}$ [°]	l [m]
D*	32727	37	15	675	8.27
TWD*	65456	138	9	300	8.10
E*	32727	129	21	422	8.06

TABLE VIII
REPLANNING EXECUTION

Alg.	t^{max}_{replan} [ms]	I^{max}_{replan}	Σt_{replan} [ms]	ΣI_{replan}
D*	17	5286	161	67457
TWD*	23	9618	419	100579
E*	14	4197	1217	135190

VI. CONCLUSION

This paper presents a comparative study of three graph based search algorithms implemented in the occupancy grid map of the environment – the D*, two-way D* and the E* algorithm. The criteria for comparison were the path characteristics, the time of execution and the number of iterations of the algorithms' main while loop. The analysis of path characteristics shows that the E* and the TWD* algorithms are more appropriate to follow by the mobile robot since they produce more natural and low cost paths than the D* algorithm. The analysis of the time of execution shows that the E* and TWD* algorithms are more computationally demanding than the D* algorithm. When changing the grid size the E* path becomes more similar to the Euclidean shortest path, while the D* and the TWD* paths remain almost the same. Using the weighted occupancy grid map with the safety cost mask has shown to be very appropriate for the E* algorithm, which calculate the shortest paths. The experiments on the real robot shows that all three algorithms have good real-time performances.

ACKNOWLEDGMENT

This research has been supported by the Ministry of Science, Education and Sports of the Republic of Croatia under grant No. $036-0363078-3018$.

REFERENCES

[1] M. Dakulovic and I. Petrovic. Two-way D* algorithm for path planning and replanning. *Robotics and Autonomous Systems*, 59(5):329–342, 2011.
[2] D. Ferguson and A. Stentz. Field D*: An Interpolation-Based Path Planner and Replanner. *Robotics Research: Results of the 12 th International Symposium ISRR(STAR: Springer Tracts in Advanced Robotics Series Volume 28)*, 28:239–253, 2007.
[3] D. B. Gennery. Traversability analysis and path planning for a planetary rover. *Autonomous Robots*, 6(2):131–146, 1999.
[4] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
[5] O. Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.
[6] J.C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Dodrecht, Netherlands, 1991.
[7] R. Philippsen and R. Siegwart. An interpolated dynamic navigation function. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3782–3789. Citeseer, 2005.
[8] S.J. Russell, P. Norvig, J.F. Canny, J. Malik, and D.D. Edwards. *Artificial intelligence: a modern approach*. Prentice Hall Englewood Cliffs, NJ, 1995.
[9] M. Seder, K. Maček, and I. Petrović. An integrated approach to real-time mobile robot control in partially known indoor environments. *Industrial Electronics Society, 2005. IECON 2005. 32nd Annual Conference of IEEE*, pages 1785–1790, 2005.
[10] M. Seder and I. Petrović. Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. *Robotics and Automation, 2007 IEEE International Conference on*, pages 1986–1991, 2007.
[11] JA Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996.
[12] A. Stentz. Optimal and efficient path planning for partially-known environments. *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3310–3317, 1994.
[13] A. Stentz. The focussed D* algorithm for real-time replanning. In *International Joint Conference on Artificial Intelligence*, volume 14, pages 1652–1659, 1995.
[14] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. Cambridge, Massachusetts: MIT Press, 2005.
[15] CM Witkowski. A parallel processor algorithm for robot route planning. *Int. Joint Conf. on Artificial Intelligence (IJCAI), Karlsruhe, West Germany*, pages 827–829, 1983.