Path Planning for Active SLAM Based on the D* Algorithm With Negative Edge Weights

Ivan Maurović, *Member, IEEE*, Marija Seder, *Member, IEEE*, Kruno Lenac, *Member, IEEE*, and Ivan Petrović, *Member, IEEE*

***Abstract*—In this paper, the problem of path planning for active simultaneous localization and mapping (SLAM) is addressed. In order to improve its localization accuracy while autonomously exploring an unknown environment the robot needs to revisit positions seen before. To that end, we propose a path planning algorithm for active SLAM that continuously improves robot's localization while moving smoothly, without stopping, toward a goal position. The algorithm is based on the D* shortest path graph search algorithm with negative edge weights for finding the shortest path taking into account localization uncertainty. The proposed path planning algorithm is suitable for exploration of highly dynamic environments with moving obstacles and dynamic changes in localization demands. While the algorithm operation is illustrated in simulation experiments, its effectiveness is verified experimentally in real-world scenarios.**

***Index Terms*—Active SLAM, dynamic environment, exploration, negative edge weight in a graph, path planning, simultaneous localization and mapping (SLAM).**

I. Introduction

The simultaneous localization and mapping (SLAM) problem has been widely researched in the last two decades and many different variations exist. Considering SLAM as a passive system the robot's path is not chosen by taking into account the localization uncertainty improvement. Thus, SLAM algorithm does not control the robot motion in any sense. If SLAM algorithm steers the robot to the areas which lower the pose uncertainty it becomes the active SLAM algorithm. In general, the idea is that the robot chooses a path which maximizes the information gain by moving to the areas already visited from the previous navigation actions where it can take measurements of already known landmarks (objects in the environment used for localization like points, lines, planes, and others). This is called loop closing in SLAM. Active SLAM minimizes localization and map uncertainty. It is always combined with a higher priority module which sets the goal positions for the robot. Very often it is the exploration module where the robot builds a map of an unknown environment by visiting unknown areas what increases localization uncertainty since there are no landmarks and areas visited before [1]–[4]. We consider a moving sensor for exploration (sensor placement problem) where the sensor is mounted on a moving robot base, unlike static sensors for localization in indoor environments [5], or exploration performed with a sensor mounted on a manipulator with a fixed base [6].

Active SLAM solutions are implemented by minimizing a certain criterion consisting of information gain. In [1], the next velocity input applied to the robot is obtained from a criterion which combines localization uncertainty and the amount of unexplored area in the next robot step. The authors use extended Kalman filter (EKF) as a basic method. A variant of model predictive control (MPC) with EKF where the robot velocity inputs are obtained as a multi step optimization for active SLAM is demonstrated in [7]. The landmarks for the localization are fixed on the optimization horizon meaning that no new landmarks are expected while the robot is moving. The problem with prediction in active SLAM is that we do not have future measurements prediction. The authors assumed that innovation in Kalman filter is Gaussian with zero mean. Simulation results show a comparison between one step and multi step ahead optimization and a benefit of the later one. Leung *et al.* [2] incorporated exploration strategy in active SLAM using MPC where the exploration areas of interest are included as artificial landmarks for SLAM what forces the active SLAM path planning to explore the environment. MPC formulation of active SLAM for line-feature-based SLAM can be found in [8]. Dellaert and Kaess [9] used iSAM instead of EKF SLAM. Kontitsis *et al.* [10] introduced an interesting idea of multiple robots active SLAM with relative entropy optimization where the cost function consists of the trace of the covariance matrix at the terminal state as a measure of the uncertainty. From the space of trajectory samples the optimal one is chosen. The EKF-based SLAM is used as a simulation example. However, no real-time applicability discussion is given. Carrillo *et al.* [11] discussed a cost function used as a measure of the localization uncertainty.

If the mobile robot path is planned only a few steps ahead there is no guaranty that the robot will reach the goal position. There are also papers dealing with a complete path of the robot. Stachniss *et al.* [12] used Rao–Blackwellized particle filter to solve SLAM problem with the exploration task.

Manuscript received November 28, 2016; accepted February 1, 2017. This work was supported in part by the Unity through Knowledge Fund (Cooperative Cloud-Based Simultaneous Localization and Mapping in Dynamic Environments) under Grant 25/15, and in part by the Ministry of Science, Education and Sports of the Republic of Croatia (Centre of Research Excellence for Data Science and Cooperative Systems) under Grant 533-19-15-0007. This paper was recommended by Associate Editor S. Nahavandi.

The authors are with the Faculty of Electrical Engineering and Computing, Department of Control and Computer Engineering, University of Zagreb, 10000 Zagreb, Croatia (e-mail: ivan.maurovic@fer.hr; marija.seder@fer.hr; kruno.lenac@fer.hr; ivan.petrovic@fer.hr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2017.2668603

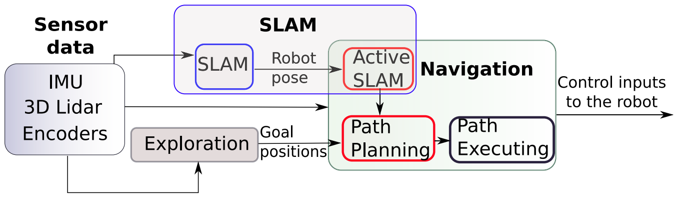

Fig. 1. Overall scheme of exploration incorporating active SLAM.

Defining exploration and place revisiting goal positions they evaluate the path to each of the goal position by estimating the change of the entropy of the particle filter caused by moving the robot to a certain goal position. Sim and Roy [13] introduced a global path planning based on the robot's position variance minimization using breadth first search over all robot positions. However, the exploration task might not be optimal as there is no exhaustive search of the space of all trajectories. Additionally, approaches where each of the potential trajectories from the robot position to the set of goal positions (specific for exploration tasks) are evaluated and the best trajectory according to SLAM criterion is chosen usually have a limited trajectory space from which the best trajectory comes from.

In some active SLAM approaches uncertainty improvement is considered when the uncertainty of the robot pose is above a certain threshold [14], [15]. Since SLAM is often combined with exploration, in these methods the robot movements switch between exploration and SLAM tasks in a discrete way depending on the uncertainty criterion value. Active SLAM problem with multiple robots is considered, where Pham and Juang [16] used predefined threshold for the switching between exploration and localization improvement. The multiple robot active SLAM is also used in [17]. A procedure of taking a fixed threshold for the loop closing is limited since it stops the robot from accomplishing the main task and control the robot to the loop closing position causing time delay in the main task. The opportunity for loop closing perhaps was better in some previous robot positions when the robot did not have a problem with localization uncertainty but the detour could have been much shorter and smoother. On the other hand, setting a subgoal for the localization would cause double path planning: from the robot position to the subgoal; from the subgoal to the goal, what is not preferable due to computation and time demands of the main task.

All mentioned methods suffer from the problem of replanning a trajectory/path in the scenario with moving obstacles where the robot needs to quit executing the planned trajectory and replan a new one in real time. In approaches like in [1], [2], and [7] the robot's path is only locally optimal where the path is planned only few steps ahead. The advantage of here proposed algorithm is a complete path from the robot position to the goal position. Active SLAM approach presented in [10] do not consider obstacles while the robot is moving, i.e., it is necessary to calculate the path from the beginning if obstacles appear, unlike in our approach where the path is efficiently recalculated in the presence of obstacles. Other common approach to active SLAM includes switching between exploration and SLAM like in [12], [14], and [15] what would interrupt the robot motion toward a goal in order to improve localization. To address this problem we propose a path planning algorithm that is able to continuously improve localization without interrupting the main task. Together with fast trajectory replanning in dynamic environment it represents the main contribution of this paper. The algorithm is based on the modified D* path planning algorithm [18]. Here it is adopted for use with negative edge weights and negative cycles in a graph. Additionally, the path planning algorithm can replan the shortest path in case of change of the localization demands.

II. Overall Problem Formulation

The active SLAM is often combined with exploration module which gives a goal position where to go next. The map of the environment is unknown in the beginning and the robot is given the task to explore the environment. Typical exploration problem incorporating SLAM, path planning and path executing is shown in Fig. 1. Connections between modules are represented with the arrows. The sensors mounted on the robot provide information about the robot and the environment like obstacles and odometry measurements. Sensor used for the exploration is a 3-D lidar mounted on top of the autonomous mobile robot platform measuring distance to obstacles in the environment. The same sensor is also used for SLAM, together with inertial measurement unit (IMU) and encoders. In our setup we also used a 2-D laser range sensor mounted in front for obstacle avoidance. Based on the sensor data SLAM module builds a map and localize the robot inside the map while exploration module calculates the next goal position. SLAM module consists of two submodules: one is SLAM, giving the robot pose in the map according to sensors' data and the other is active SLAM giving a discrete set of points that should be preferable to visit in order to increase the localization uncertainty. Navigation module generates robot trajectories which take into account exploration regarding the goal position and SLAM demands regarding preferable localization points. It consists of the path planning and path executing part responsible for generating robot trajectories. Active SLAM is also considered as a part of navigation module since it can redirect the robot path into the localization preferable area. The output signal from the navigation module is calculated by the path executing module by which the robot follows the given path and takes into account the model of the robot. The proposed contribution—path planning for active SLAM algorithm—is affecting active SLAM and path planning module colored red in the figure. The rest of the modules are briefly described bellow.

A. Exploration

The robot starts with an unknown map of the environment and takes an initial laser scan. Based on the first laser scan it builds an initial polygonal map of the environment and calculates the next positions from where to take the next scan to maximize the amount of unexplored area which can be seen from the next scan. The exploration is finished when the whole environment is covered by the laser sensor. The algorithm gives a goal positions in front of the lines which divide known and unknown areas. In our previous work, we tested the algorithm on a complex 3-D environment. Details of the exploration method can be found in [19].

B. SLAM

We have used graph-based SLAM algorithm developed within our group [20]. It is based on exactly sparse delayed state filter (ESDSF) [21]. ESDSF is used for estimation of Gaussian vehicle's trajectory X which consists of n pose samples $X_i = [q_i \ t_i]^{\mathrm{T}}, \ i = 0, \dots, n-1$

$$X = \begin{pmatrix} X_0 \\ X_1 \\ \cdot \\ \cdot \\ \cdot \\ X_{n-1} \end{pmatrix}, \ X \sim \ \mathcal{N}(\mu, \Sigma) = \mathcal{N}^{-1}(\eta, \Lambda) \qquad (1)$$

where q_i represents mobile robot's orientation in the global frame in the form of quaternion and t_i its position in the frame. The relation between μ and η is: $\eta = \Lambda\mu$, and between covariance $\sum$ and information matrix Λ: $\Lambda = \sum^{-1}$. Since our robot moved through one building with flat floor for the simplicity we used 2-D motion model ($t_i = [x_i, y_i]$, $q_i = [s_i, q_{3,i}]$) described as nonlinear first order Markov process

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \\ s_{n+1} \\ q_{3,n+1} \end{pmatrix} = \begin{pmatrix} x_n + \Delta x \\ y_n + \Delta y \\ \frac{s_n - \frac{1}{2} q_{3,n} \Delta y}{\sqrt{1 + \frac{\Delta\gamma^2}{4}}} \\ \frac{q_{3,n} + \frac{s_n \Delta\gamma}{2}}{\sqrt{1 + \frac{\Delta\gamma^2}{4}}} \end{pmatrix} + w_n \qquad (2)$$

where Δx, Δy, and $\Delta\gamma$ are changes in robots position and orientation in every step obtained from IMU and wheel encoders measurements and w_n is the white noise with mean value 0 and the covariance Q.

As the robot moves through the environment new states are added to the trajectory X when the pose difference f_c between the last added state X_{n-1} and the current robot pose X_n is larger than the predefined threshold. Cost function f_c takes into account both angle and distance differences between two states X_i, X_j and is calculated as

$$f_c(i,j) = d\big(X_i, X_j\big) + \alpha\big|\Delta_{\mathrm{Yaw}}\big(X_i, X_j\big)\big| \qquad (3)$$

where $d(X_i, X_j)$ is the Euclidean distance between the states X_i and X_j, and Δ_{Yaw} is the difference of the angle between those states and α is a scaling factor. Whenever a new state is added, the current measurements from the robot's sensors are recorded and associated with the state. Update of the trajectory occurs by calculating relative pose between two states from the associated measurements. In order to calculate the relative pose between two states the associated measurements need to contain enough similar landmarks which means they need to be taken from approximately the same location. This is why for a successful trajectory update the robot has to visit already traversed areas and achieve loop closing. The question is in which state should the loop closing points be placed. Although every loop closing would increase accuracy of the trajectory, SLAM update and the afterwards global map update are performance costly operations that should not be executed if the impact on the accuracy is too small. Moreover in order to close the loop, the robot will need to diverse from the shortest path planned to the exploration goal. This is why loop closing points should be placed only in states in which loop closing with high enough impact on the overall trajectory accuracy can be achieved.

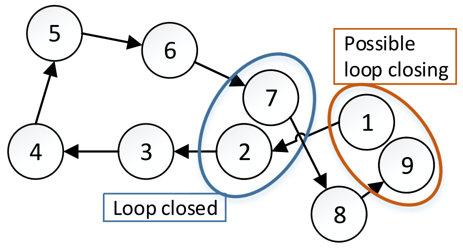

Fig. 2. Topological distance as a measure of information gain.

Whenever a new state is added to the trajectory X, the active SLAM algorithm begins to search for possible loop closings between newly added state X_i and all other states X_j, $j \neq i$, in the trajectory. The state X_j is chosen for the loop closing if it satisfies two conditions. First condition is that the Euclidean distance from X_i has to be lower than a predefined value $d_{\min}$. This condition imposes high probability of registration between states X_j and X_i since it is likely that there are enough similar landmarks (planar surface segments) between local maps assigned to them. Second condition that the state X_j has to satisfy, in order to be chosen for the loop closing is that the resulting SLAM update will have high enough impact on the trajectory accuracy. In general, update impact on the trajectory accuracy is proportional to the sum J of the cost functions f_c between all neighboring states in the trajectory moving from the state X_j to the state X_i. The sum of cost functions between states X_j and X_i is calculated as

$$J = \sum_{m=j}^{i-1} f_c(m, m+1). \qquad (4)$$

However, the problem with using sum of all cost functions between neighboring states to calculate the trajectory accuracy impact of the loop closing, arises if there were previous loop closing detections. For example, let us consider situation illustrated in Fig. 2, where we want to measure the accuracy impact of loop closing between states X_1 and X_9. Although the sum of cost functions between all states from X_9 to X_1 is high, since an update occurred between states X_7 and X_2, the overall impact on the trajectory accuracy from closing loop between states X_9 and X_1 is small. This is because a lot of information that would be gained from loop closing between states X_1 and X_9 was already gained by loop closing between states X_2 and X_7. This is why we use approach similar to [14] as a measure of accuracy impact of loop closing. Accuracy impact is determined using topological distance. Topological distance is calculated from the graph ${}^{T}g$ generated from the pose graph incidence matrix ${}^{T}I$. Pose graph incidence matrix ${}^{T}I$ is $n \times n$ matrix, where n is the number of states in the trajectory, whose elements are given by

$$ {}^{T}I_{ij} = \begin{cases} 1, & \text{if state } X_i \text{ is connected with state } X_j \\ 0, & \text{otherwise.} \end{cases} \qquad (5) $$

States are connected if they are neighboring states (i.e., state X_i is connected with states X_{i+1} and X_{i-1}) or if the pose constraint was formed between them (i.e., loop closing was detected and trajectory update executed). Nodes in the graph ${}^{T}g$ are represented by the states and connections between nodes i and j exist if the element (i,j) in matrix ${}^{T}I$ is 1. Weight of the connection $w_{i,j}$ between states i and j is

$$ w_{i,j} = \begin{cases} f_c(i,j), & \text{if } |i-j| = 1 \\ 0, & \text{otherwise} \end{cases} \qquad (6) $$

this ensures that connections made by the previous loop closing have 0 weight and will not increase measured topological distance between two states. Topological distance between states (X_i, X_j) is calculated as the shortest path from the node i to the node j in the graph ${}^{T}g$. The shortest path is calculated using the A* algorithm. For example, topological distance from the nodes 1–9 (Fig. 2) would be the sum of cost functions between the states (X_1, X_2), (X_7, X_8), and (X_8, X_9). Now when we can calculate topological distance between two states (X_i, X_j), we can form the second condition that states have to satisfy in order to be chosen for the loop closing

$$ {}^{T}d\big(X_i, X_j\big) > D_{\max} \qquad (7) $$

where ${}^{T}d$ is the topological distance between the states X_i and X_j, and $D_{\max}$ is a predefined topological distance threshold. For all states that satisfy both conditions, the state with the highest topological distance from X_i is selected for loop closing.

1) Path Planning: When the robot has all information about the static and dynamic obstacles in its surrounding it obtains a shortest, obstacle free path to reach the goal. In the proposed method the additional input to the path planning module are loop closing points from the active SLAM module. If the cost of the detour the robot takes to reach localization points is acceptable according to localization accuracy increase, the path will be planned through the localization point. The path planning module is based on the D* path planning algorithm and it is the main contribution of this paper. It is explained in detail in the next section.

2) Path Executing: Path executing module receives planned path from the path planning module and brings signals to control the robot's wheels to reach the goal position. The outputs are linear and angular velocities of the robot in each time step. We used our receding horizon control approach for the trajectory execution based on the planned D* path [22].

III. D* With Negative Edge Weight Path Planning

There has been a large amount of approaches on planning the robot trajectories, path planning, and path following [23], [24]. In this paper, we use a graph representation of the environment as a collection of nodes and edges, $\mathcal{G}(\mathcal{N}, \mathcal{E}, \mathcal{W})$ with $\mathcal{N}$ representing nodes and $\mathcal{E}$ representing edges and $\mathcal{W}$ referring to edge weights. The environment is represented by a discrete grid where each grid cell is occupied or empty, representing nodes $\mathcal{N}$ in the graph. Such a grid is called the occupancy grid map created by approximate cell decomposition of the environment [25], [26]. The current robot position is denoted with $\boldsymbol{x} = [x\, y\, \theta]^T$.

Besides standard cells in a grid map with occupied and empty status for the purpose of active SLAM we add the localization improvement cells $\mathcal{L} \subset \mathcal{N}$ as a loop closing points from active SLAM and consider them as a special status of the cell.

Search for a shortest path in a graph uses the criterion as a sum of all edge weights from the robot position to the goal position. The edge weight of the empty, i.e., the cost of entering to an empty cell is set to the Euclidean distance between two adjacent cells depending on the cell size while the edge weight of an occupied cell is set to ∞. The localization cells should attract the robot to go through the localization cell thus the edge weight or the cost to enter the localization cells has to be lower than the other cells in the grid. Setting the edge weight to 0 is not giving anything to attract the robot. Actually the edge weight needs to have a negative value. The edge weight of the localization cell compensates a detour the robot takes from the shortest path on the way to the goal position what can be done only using negative cell edge weight.

A standard graph search in robotics like A^*, D^* or simple Dijkstra algorithm cannot be used since we have negative edge weights in the graph. Adding a negative edge weight cell in the graph causes problems with finding a shortest path using common graph search algorithms in robotics, especially since there are also negative cycles where shortest path does not exist. The shortest path problem with the negative edge weights is used in [27], where the problem is solved using dynamic programming Bellman–Ford algorithm for the shortest path planning. Bellman–Ford algorithm can solve the problem with the negative edge weights but has higher complexity than Dijkstra and it is not suitable for dynamic environment [28]. If something changes in the environment or dynamic obstacle appears the whole computational process has to be repeated. The negative cycle problem with the shortest path still remains since Bellman–Ford can only detect a negative cycle but it does not solve it in any way. Therefore, we proposed an extension of the D^* algorithm which solves the shortest path problem with negative edge weights.

A. Graph Creation and Search

Weighted undirected graph $\mathcal{G}(\mathcal{N}, \mathcal{E}, \mathcal{W})$ is created in the occupancy grid map. Two nodes $i, j \in \mathcal{N}$ in the graph are *neighbors* if corresponding Cartesian coordinates of cell centers $c_i, c_j \in \mathbb{R}^2$ have L_∞ distance equal to the length of the cell width e_{cell}, i.e., $\|c_i - c_j\|_\infty = e_{\text{cell}}$. The set of edges is defined as $\mathcal{E} = \{e_{i,j} := \{i, j\} \mid i, j \in \mathcal{N}, i \text{ and } j \text{ are neighbors}\}$. 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.

In standard binary occupancy grid maps there are two values of transitions between neighbors: straight and diagonal transition what could present real Euclidean distance, i.e.,

$w_{i,j} = \|c_i - c_j\|_2$. For the purpose of active SLAM negative cost transitions between the negative cell from $\mathcal{L}$ and a free cell from $\mathcal{N} \setminus \mathcal{L}$ is introduced in the graph. The negative value is a function of the expected localization gain from the SLAM update at a certain negative cell position. For the problem, we solve the edge weight has the following structure:

$$w_{i,j} = \lambda \left\| c_i - c_j \right\|_2 \tag{8}$$

where $\lambda = 1$ for empty cells, ∞ for occupied cells and $\lambda < 0$ for localization (negative) cells. For the negative values, λ is a function of a topological distance ${}^{T}d(X_n, X_j)$ between the current robot's state X_n in SLAM and the state X_j in SLAM. State X_j in SLAM is represented with the negative cell in the graph

$$\lambda = f\left({}^{T}d\left(X_n, X_j\right)\right). \tag{9}$$

Assume that the start node and the goal node are specified in the graph $\mathcal{G}(\mathcal{N}, \mathcal{E}, \mathcal{W})$. The search for the shortest path from the start node to the goal node has to be performed. A path $\mathcal{P} = \mathcal{P}(\text{start, goal})$ is in the graph $\mathcal{G}(\mathcal{N}, \mathcal{E}, \mathcal{W})$ is defined as

$$\begin{aligned} \mathcal{P}[1] &= \text{start}, \mathcal{P}[|\mathcal{P}|] = \text{goal} \\ \mathcal{P}[i] &\in \mathcal{N}, i = 1, \ldots, |\mathcal{P}| \\ \left\{\mathcal{P}[j], \mathcal{P}\left[j+1\right]\right\} &\in \mathcal{E}, j = 1, \ldots, |\mathcal{P}| - 1 \\ \mathcal{P}[i] &\neq \mathcal{P}[j], \quad i, j = 1, \ldots, |\mathcal{P}|, i \neq j \end{aligned} \tag{10}$$

where $|\mathcal{P}|$ represents the path length in the number of cells.

The cost of the path $\mathcal{P}$ is defined as the sum of weights of edges along the path, that is

$$h(\mathcal{P}) := \sum_{i=1}^{|\mathcal{P}|-1} w_{\mathcal{P}[i],\mathcal{P}[i+1]}. \tag{11}$$

Let $\pi(\text{start, goal})$ be the set of all possible paths between the start node and the goal node in the graph $\mathcal{G}(\mathcal{N}, \mathcal{E}, \mathcal{W})$. The *optimal cost* of the path from the start node to the goal node that has to be searched for is defined as

$$\begin{aligned} h^*(\text{start, goal}) := \min_{\mathcal{P}} \;& h(\mathcal{P}) \\ \text{s.t.} \;& \mathcal{P} \in \pi(\text{start, goal}) \end{aligned} \tag{12}$$

with implicit assumption that $h^*(\text{start, goal}) = \infty$ if $\pi(\text{start, goal}) = \emptyset$. The shortest path from the start node to the goal node is the path (or more than one path) that has the optimal cost $h^*(\text{start, goal})$.

B. D* Algorithm

The D* algorithm is a well known graph search algorithm capable of fast replanning in dynamic environments [18]. It is also known as the dynamic version of the Dijkstra's algorithm or dynamic version of the A* algorithm without the heuristic function [23]. The D* algorithm finds the shortest path in graphs in which weights change during the time, i.e., occupancy values become higher or lower due to obstacles movement.

For every searched node Z, the D* algorithm computes the cost value $h(Z)$ of the path from the node Z to the goal node and the value of the key function $k(Z)$ for the replanning process, which stores the old values $h(Z)$ before changes of weights in the graph happened.

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 ($R = \text{start}$) and replanning is performed if the robot detects nodes with changed occupancy values during its motion. D* uses the set OPEN as a temporary storage for the currently examined nodes. Details can be found in [18].

C. Negative Edge Weight D* Algorithm—ND* Algorithm

The proposed algorithm is given in Algorithm 1 where the main differences from the original D* algorithm are highlighted and explained in this section. We named the proposed graph search algorithm the negative edge weight D* algorithm (ND* algorithm) the main D* function is renamed as PROCESS-STATE-NEG-EDGE().

In PROCESS-STATE-NEG-EDGE() algorithm lines 5–7 are different from the standard D* algorithm. Before the initial search the set of all graph nodes with negative edge weight is created, called NEGATIVE_CELL.

The algorithm starts by adding the goal node to the OPEN list. Every node Z has associated tag $t(Z)$ with a value NEW if the node has never been on the open list, OPEN if it is currently on the OPEN list and CLOSED if the node was on the OPEN list. In the initial step $t(Z)$ for all nodes is set to NEW. PROCESS-STATE-NEG-EDGE() function is invoked repeatedly until OPEN list is empty. The rest of the embedded functions are MIN-STATE() which returns the node with minimum k value on the OPEN list, DELETE(Z) which removes the node from the OPEN list and INSERT(Z,h(Z)) which inserts the node Z on the OPEN list and computes h(Z). INSERT(Z,h(Z)) is given in Algorithm 2. At the end of initial path planning process every node has a parent node, i.e., $b(Y) = Z$, where Y is a parent node of Z or we can say Y is pointing at Z. The shortest path for the node Y is obtained following pointers through parents nodes to the goal node.

Executing CREATE-BLACKLIST() in Algorithm 1 enables handling negative edge weights in the graph. If the negative cell is on the OPEN list and has minimum k value among the other cells on the list, the BLACKLIST is created by the following backpointers starting from the negative edge weight node. Creating the BLACKLIST on a such way it actually represents the shortest path from the negative edge weight cell to the goal node. In the standard D* algorithm with no negative cells the initial step is the Dijkstra shortest path search. Only line segment from 15–19 is executed. Dijkstra algorithm does not work with negative edge weights, i.e., does not guaranties optimality with negative cells. In the proposed algorithm when condition Z $\in$ NEGATIVE_CELL is true the path from the negative cell to the goal position is the shortest path. We use that path fixed during the rest of the ND* search. Creating BLACKLIST we do not allow nodes on the list to change the pointer while the rest of nodes will be changed and redirected according to negative cell node. By this we prevent the loop cycles, or the negative cycles in the graph.

Algorithm 1 PROCESS−STATE−NEG−EDGE()

```
1:  Z=MIN−STATE(O)
2:  if Z=NULL then
3:      return -1
4:  end if
5:  if Z ∈ NEGATIVE_CELL then
6:      CREATE−BLACKLIST()
7:  end if
8:  $k_{old}$=GET−KMIN(); DELETE (Z)
9:  if $k_{old} < h(Z)$ then
10:     for each neighbor Y of Z
11:     if $h(Y) \leq k_{old}$ and $h(Z) > h(Y) + w_{Y,Z}$ and (Y ∉ BLACKLIST) then
12:         $b(Z) = Y$; $h(Z) = h(Y) + w_{Y,Z}$
13:     end if
14: end if
15: if $k_{old} = h(Z)$ then
16:     for each neighbor $Y$ of $Z$:
17:     if $t(Y) = NEW$ or $(b(Y) = Z$ and $h(Y) \neq h(Z) + w_{Z,Y})$ or $(b(Y) \neq Z$ and $h(Y) > h(Z) + w_{Z,Y})$ and (Y ∉ BLACKLIST) then
18:         $b(Y) = Z$; INSERT$(Y, h(Z) + w_{Z,Y})$
19:     end if
20: else
21:     for each neighbor $Y$ of $Z$
22:     if $t(Y) = NEW$ or $(b(Y) = Z$ and $h(Y) \neq h(Z) + w_{Z,Y})$ and (Y ∉ BLACKLIST) then
23:         $b(Y) = Z$; INSERT$(Y, h(Z) + w_{Z,Y})$
24:     else if $b(Y) \neq Z$ and $h(Y) > h(Z) + w_{Z,Y}$ and (Y ∉ BLACKLIST) then
25:         INSERT$(Z, h(Z))$
26:     else if $b(Y) \neq Z$ and $h(Z) > h(Y) + w_{Y,Z}$ and $t(Y) = CLOSED$ and $h(Y) > k_{old}$ and (Y ∉ BLACKLIST) then
27:         INSERT$(Y, h(Y))$
28:     end if
29: end if
```

When initial phase calculates the shortest path from each node to the goal the robot starts to execute the path following task. When the change of the map is detected with sensors on the robot the function MAP-CHANGE() is called. The part of the proposed algorithm is given in Algorithm 3. Map change could be caused by a dynamic obstacle, creating a new localization point or removing already visited localization point. The function MAP-CHANGE() changes edge weights and enters affected cell on the open list. The difference from the standard D* is that it also checks if the modification in the map is on the shortest path from the negative cell to the goal position, i.e., on the BLACKLIST. If that happens all cells with the shortest path going through the BLACKLIST are reset by invoking RESET_BLACKLIST(). Status of the cell is set to NEW, their h value is set to obstacle and pointers are deleted. Also, all cells on the BLACKLIST are set on the OPEN list while deleting the BLACKLIST.

Algorithm 2 INSERT(Z,h_n)

```
1: if t(Z)=NEW then
2:     k(Z)=$h_n$
3: else if t(Z)=OPEN then
4:     k(Z)=min(k(Z),$h_n$)
5: else if t(Z)=CLOSED then
6:     k(Z)=min(h(Z),$h_n$)
7: end if
8: h(Z)=$h_n$
9: t(Z)=OPEN
```

Algorithm 3 MAP−CHANGE(Z,w_val)

```
1: for each neighbor $Y$ of $Z$ do
2:     $w_{Z,Y}$=w_val
3: end for
4: if t(Z)=CLOSED then
5:     INSERT(Z,h(Z))
6: end if
7: if Z ∈ BLACK_LIST then
8:     RESET_BLACKLIST()
9: end if
```

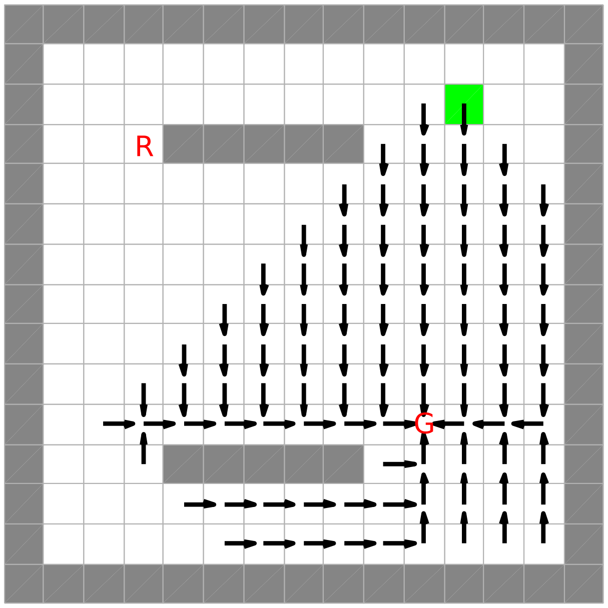

Fig. 3. Beginning of initial step.

D. Illustration of Operation

In this section, we illustrate how the algorithm operates in a simple environment. Starting from the initial step where the map of the environment and the robot's goal position is given. The algorithm starts by putting a goal node on the open list and expanding it with its neighbor cells. The number of neighbor cells depends on a realization of the graph from the occupancy grid map. Here we use four neighbors due to simplicity but the algorithm works independently from the number of neighbors. Fig. 3 represents the map showing pointers at the moment when the negative cell has been expanded and

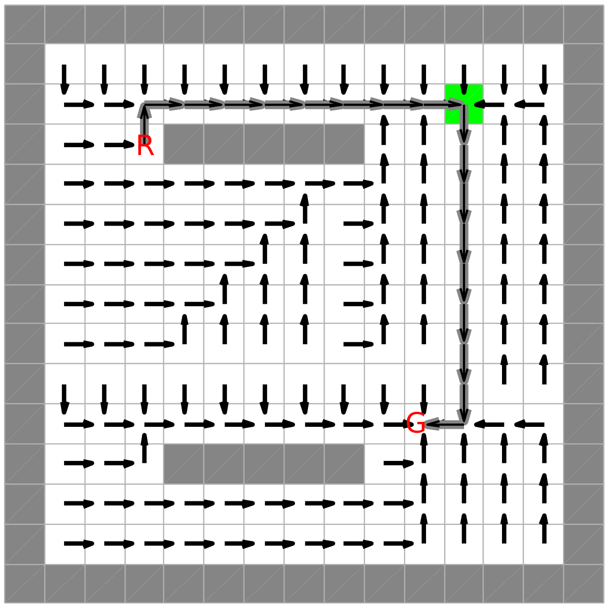

Fig. 4. Final path planning of the initial step.

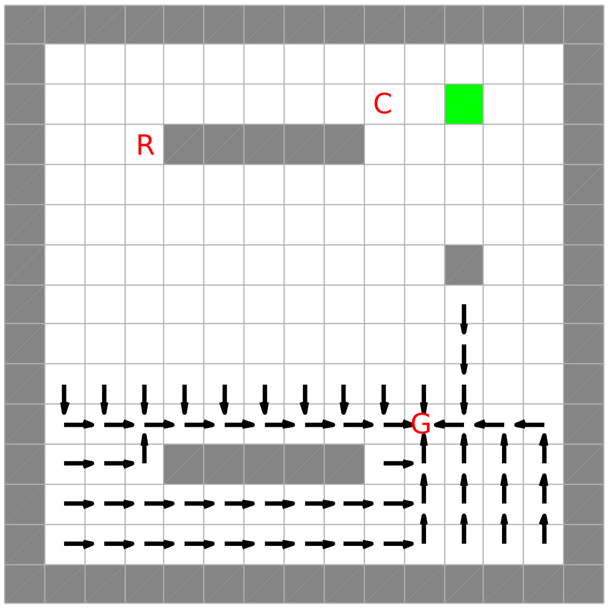

Fig. 5. Reset the map after obstacles on the blacklist.

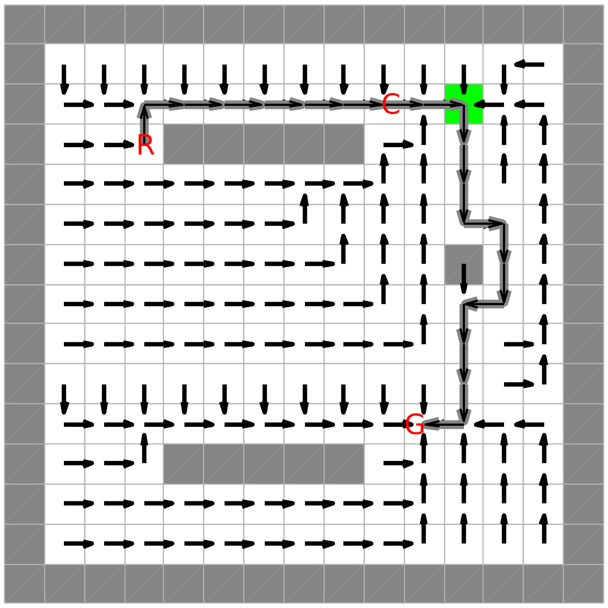

Fig. 6. Path planning with an obstacle on the blacklist.

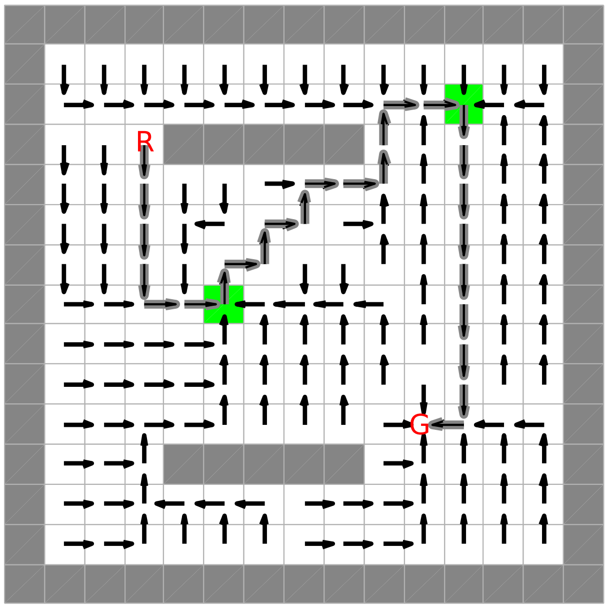

Fig. 7. Two negative cells in the map.

removed from the open list. The robot's start and goal position are denoted with R and G, respectively. The obstacles are colored gray while negative edge weight cells are green. Each cell which has been examined at that moment would have the shortest path if there was no negative cell in the map. When the algorithm reaches the negative cell (line 5) it activates and generates the blacklist to fix the shortest path from the negative cell to the goal position. The pointers from the blacklist remain unchanged during the path planning while the other cells change the pointers after the negative cell is detected. This way the graph cycles are avoided. Line 17 in Algorithm 1 with condition $b(Y) \neq Z$ AND $h(Y) > h(Z)+c(Z, Y)$ activates and redirect the cells. This is not the case with the standard D* algorithm where in the initial step the cells with CLOSED status already have the shortest path and are not considered. In Fig. 4, the final, initial path planning with the shortest path with thicker arrows is shown. As it can be seen in Fig. 4 all cells in the upper part of the map are attracted by the negative cell. For the simulation purpose negative edge weight is set to -15 where the other cells' edge weight is set to 1.

When the robot has the shortest path it starts to follow the path. If nothing changes on the way the whole pointer structure remains the same and the robot reaches the goal. Since the dynamic environment is highly expected where the robot has any kind of interaction with humans or other moving obstacles the algorithm begins with replanning phase when a change in the environment is detected. Three typical map change scenarios are possible. When a dynamic obstacle is in the map area

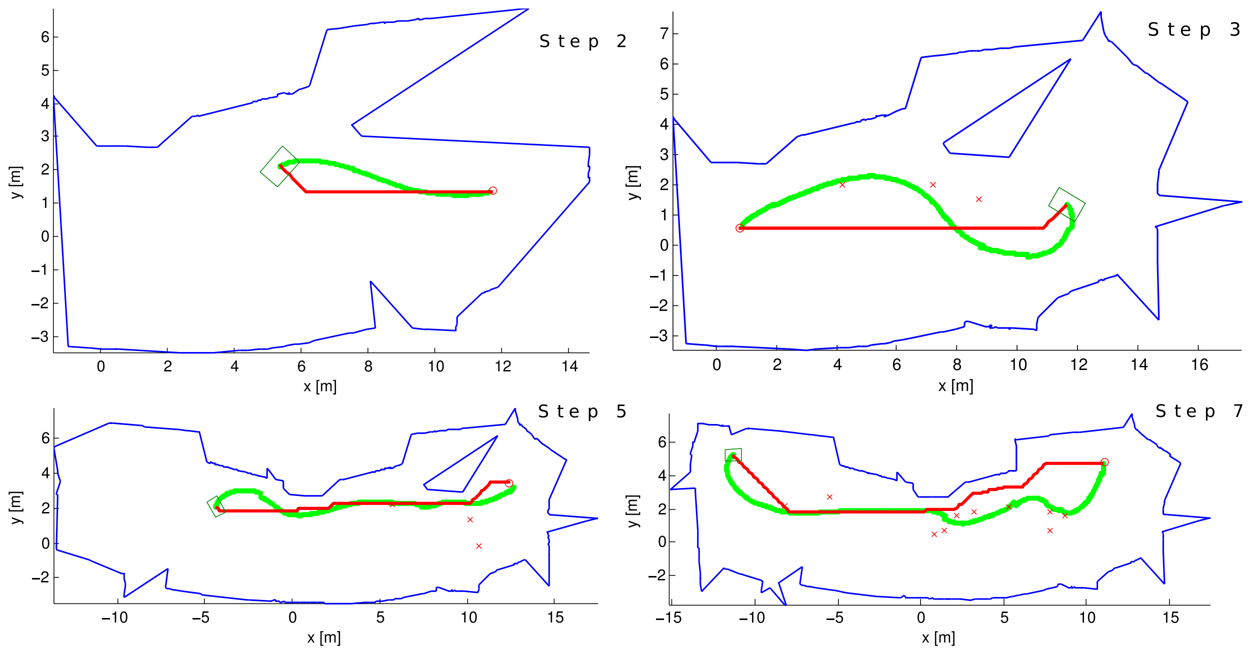

Fig. 8. Experimental results—exploration with the loop closing. Red line is the planned robot path, green line is the executed path and red *x* are the localization points.

not affected by the negative cell or cells on the blacklist the algorithm performs as it is expected from standard D* algorithm. The second scenario appears when the obstacle is on the blacklist which forces the algorithm to allow changes on the blacklist and the third scenario that activates replanning is when a new localization point appears or when an already existing localization point is removed from the map. The latest two are considered in the next example since the first one is standard as in the D* algorithm.

1) Obstacle on the Blacklist: When an obstacle is on the blacklist the pointers on the blacklist need to be changed what would cause cycles in the graph with negative edge weight. To avoid cycles in the graph we implemented a special care for that case in the function MAP-CHANGE() in line 7. Calling RESET_BLACKLIST() resets only cells for which the shortest path goes through the blacklist, i.e., cells affected by the obstacle on the blacklist. Fig. 5 shows the map after MAP-CHANGE() in line 8 execution. The robot starts moving and at the position denoted with C detects the obstacle on the blacklist. It can be seen that complete upper part of the map is reset since the negative edge value is set to -15 and attracts relatively large area in that example map. In real scenarios affected area are usually local cells around the negative cell what is more acceptable from the algorithm complexity point of view. Path planning continue with setting the obstacle cell on the open list and invoking PROCESS-STATE-NEG-EDGE(). The result is shown in Fig. 6, where the robot position is denoted with *C*.

2) New Localization Cell: The number of localization cells considered in the map depends on the area where the robot is expected to go. Thus, when the robot moves it enters a new area where new localization cells appear. All cells with changed status are set on a OPEN list with calling MAP-CHANGE(). In order to guaranty absence of negative cycles the shortest path from the negative cell to the goal needs to be found to create the blacklist. When a new localization cell appears it already has the shortest path to the goal position due to the initial planning. Following the shortest path the blacklist is created and replanning executes invoking PROCESS-STATE-NEG-EDGE(). Some nodes redirect their pointer so that path goes through the new localization cell.

Fig. 7 shows the situation when two negative cells are active in the environment. A new negative cell edge weight is set to -10 what results with the shortest path going through the both negative cells. Note that the blacklist consists of the shortest path with both negative cells inside. In a different arrangement of the negative cell it could happen that the shortest path goes only through one cell what would create two blacklists for each of the negative cell. The same scenario happens if there are more than two negative cells. On the other hand, when the robot closes the loop the negative cell needs to be removed and not used any more. The edge weight is set to positive value and the cell is put on the OPEN list. The blacklist is deleted for that cell and PROCESS-STATE-NEG-EDGE() recalculates the path.

IV. Experimental Results

The proposed algorithm was tested experimentally on a differential drive mobile robot Husky equipped with 2-D SICK LMS100-10000 laser and 3-D Velodyne LiDAR. The main task was exploration of the environment based on the 2-D data available from the laser. The exploration algorithm used in the experiment can be found in [19] and is shortly described in Section II-A. The next robot goal positions are chosen to maximize unexplored area which can be seen from the next robot's positions. The overall system was implemented under the robot operating system.

We did experiments under various conditions including no obstacles and dynamic obstacles on the robot's path, appearing a group of the people around the robot, varying with obstacles size, etc. The negative cells were added dynamically when the robot entered in the vicinity of the loop closing points. However, not all negative cells were part of the robot's path to the goal. Fig. 8 shows a few typical exploration steps from one of the experiments. The polygon shown in each figure represents the explored environment from the beginning until the current step labeled in the figure. The robot's current position is denoted by a rectangular shape object. Two paths connecting consecutive goal points represent planned path and executed path colored red and green, respectively. The localization points are marked with red x inside the exploration polygon. The robot starts exploring the environment and makes an initial map of the explored area and chooses the next exploration goal. On the way to the next goal, what is shown in step 2, the planned and executed paths are similar. The difference in the beginning is due to path executing module which takes into account the mobile robot dynamics and initial orientation. As can be seen in step 2 there are no localization points on the way since the robot has entered a new area for the first time and the loop closing cannot be achieved if the robot has not been there before. Step 3 brings the robot back to the area visited before and shows the ND* algorithm steering the robot from the planned trajectory to lower the localization uncertainty. While the robot moves following an already planned path the localization points appear one by one since the robot was already moving in that area. The area in which we search for a loop closing position is a circle with a radius of 4 m around the robot. Negative value λ for the current robot's SLAM state X_n used in the experiment is calculated using the following equation:

$$\lambda(X) = \frac{{}^{T}d\big(X_n, X_j\big)}{-500} \tag{13}$$

where ${}^{T}d(X_n, X_j)$ refers to a topological distance for the SLAM state X_n, explained in Section II-B, while X_j is SLAM state inside 4 m radius circle. We used -500 to scale the topological distance to use it as a edge weight in the graph. The value was chosen experimentally so the topological distance of 500 means the negative cell attracts only one cell around. The new negative cell points are accepted for ND* path planning only if the negative value of a new points is smaller than the current active point or when there is no negative cell in the graph. The first point that appeared is the middle one with the negative value of -15 and the robot was attracted by the negative cell and loop was closed. As can be seen the robot did not go exactly through the marked negative point because we set the loop closing tolerance to 1 m since the robot can successfully close the loop when it is inside the circle with radius of 1 m around the loop closing point. The next localization point was the left one with the negative value of -25 and the robot also went through that negative point to improve the localization. When the loop was closed the most right loop closing points appeared with the negative edge weight of -21 but the path planning did not change the trajectory since the point was far away from the current robot position.

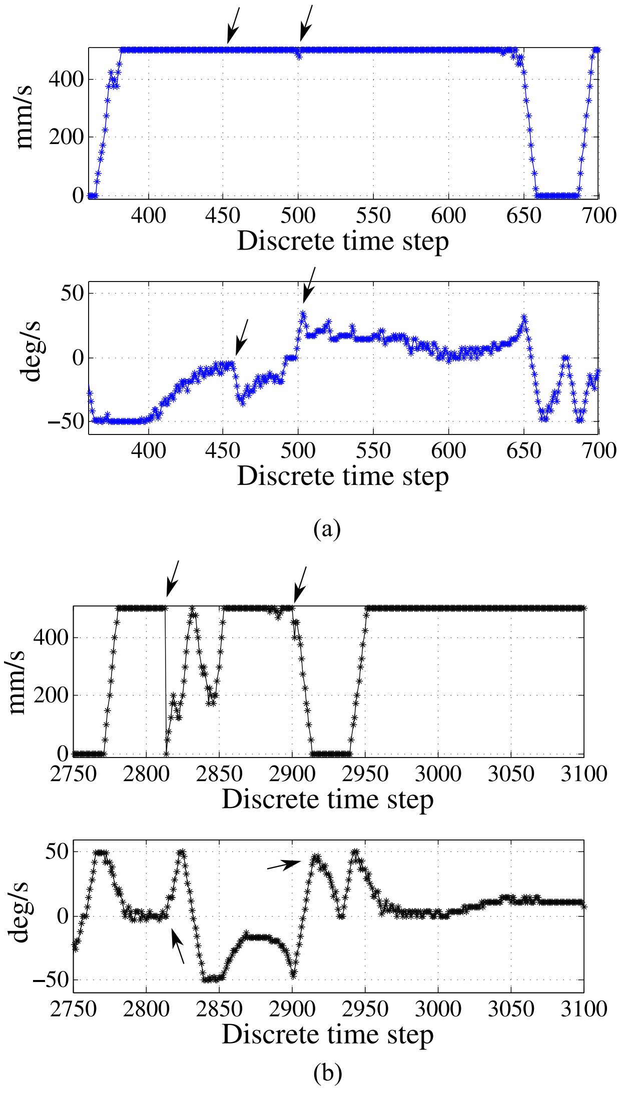

Fig. 9. Velocity comparison. (a) ND* active SLAM—linear velocity (top) and angular velocity (bottom). (b) Switching active SLAM—linear velocity (top) and angular velocity (bottom).

Steps 5 and 7 show longer paths where can be seen how the active SLAM affected robot's path according to the initially planned path. In step 5, one localization point was on the robot's initial path while the others were not chosen by the ND* algorithm as a good enough to take a detour to close the loop. Step 7 of the exploration process clearly shows deviation from the initially shortest path due to negative cells appeared close to the robot's path. The exploration algorithm finished in 13 steps and the whole environment was explored.

The algorithm was implemented on a ThinkPad P50 with Intel Core i7 2.60 GHz processor and 8 GB of RAM. The algorithm processing time is shown in Fig. 10. For each step of the ND* algorithm, i.e., for each call of the ND* algorithm, the processing time is presented in ms. It can be seen that all calculations were done within 26 ms.

We have explored the same environment with a switching active SLAM concept from our previous work described in [15]. Fig. 9 shows robot's angular and linear velocity comparison for the switching active SLAM and ND* active SLAM proposed in this paper. For both algorithms the velocities are

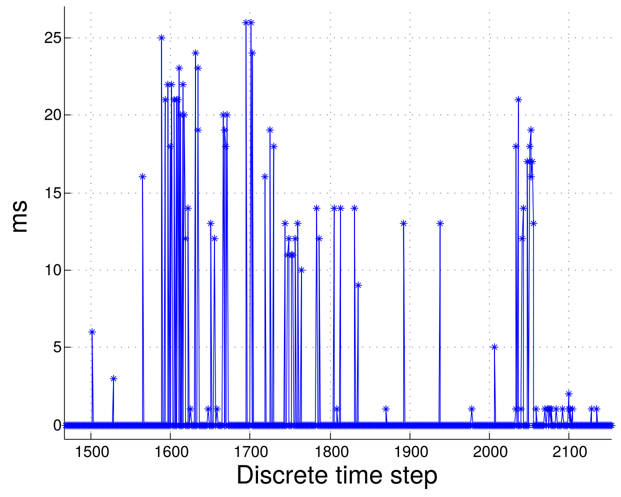

Fig. 10. Processing time for ND* algorithm.

taken at the point when the robot decides to turn from the planned path to close the loop. The arrows are pointing at the moment when the localization point appears. x-axis represents discrete time where each time step corresponds to a period of 100 ms. It can be seen that with the proposed algorithm the robot has continuous and smooth movements in comparison to the switching SLAM algorithm where the robot needs to stop, set velocity to zero and then continue toward the loop closing point. For each of the velocity profile two loop closing points appear while the robot was moving to the goal position. Besides the velocity profile difference the path planning for the switching active SLAM is done from the beginning when a new localization point appears while with ND* we have only replanning. The average velocity is higher with ND* what brings faster robot's movement and together with less time needed for path planning implies faster exploration. With ND* algorithm we had more often loop closing what keeps localization uncertainty inside boundaries unlike in switching SLAM where the localization uncertainty rapidly drops when the loop is closed.

V. Conclusion

In this paper, we presented the path planning for active SLAM loop closing. The path planning is based on the D* algorithm modification where we introduced negative edge weights in a graph for the shortest path planning. Standard graph search algorithms in robotics cannot handle negative edge weights due to negative cycles which would appear as a shortest path solution. The active SLAM solution presented in this paper can effectively deal with dynamic changes in the environment including moving obstacles and localization demands which can appear while robot is moving. Throughout a simple environment simulation it was shown how negative cycles are handled with a modification of the D* algorithm.

In the experiment we have shown in a real-world scenario with exploration task that the proposed path planning algorithm for active SLAM efficiently closes a loop planning the path which goes through already visited area. The robot's path is continuous and when a localization point appears the robot's speed does not rapidly change what gives faster robot motion and faster exploration while simultaneously improving the localization uncertainty. The shortest path is changed only locally when the dynamic obstacle or dynamic localization pose appears thus the time needed to travel to the goal position is decreased.

References

[1] Y. Liu, F. Sun, T. Tao, J. Yuan, and C. Li, "A solution to active simultaneous localization and mapping problem based on optimal control," in *Proc. Int. Conf. Mechatronics Autom. (ICMA)*, Harbin, China, 2007, pp. 314–319.

[2] C. Leung, S. Huang, and G. Dissanayake, "Active SLAM using model predictive control and attractor based exploration," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, Beijing, China, 2006, pp. 5026–5031.

[3] J. Vallvé and J. Andrade-Cetto, "Active pose SLAM with RRT*," in *Proc. IEEE Int. Conf. Robot. Autom.*, Seattle, WA, USA, 2015, pp. 2167–2173.

[4] R. Valencia, J. V. Miró, G. Dissanayake, and J. Andrade-Cetto, "Active pose SLAM," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2012, pp. 1885–1891.

[5] I. Vlasenko, I. Nikolaidis, and E. Stroulia, "The smart-condo: Optimizing sensor placement for indoor localization," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 3, pp. 436–453, Mar. 2015.

[6] S. Salan, E. Drumwright, and K.-I. Lin, "Minimum-energy robotic exploration: A formulation and an approach," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 1, pp. 175–182, Jan. 2015.

[7] S. Huang, N. M. Kwok, G. Dissanayake, Q. P. Ha, and G. Fang, "Multi-step look-ahead trajectory planning in SLAM: Possibility and necessity," in *Proc. IEEE Int. Conf. Robot. Autom.*, Barcelona, Spain, 2005, pp. 1091–1096.

[8] C. Leung, S. Huang, and G. Dissanayake, "Active SLAM in structured environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, Pasadena, CA, USA, 2008, pp. 1898–1903.

[9] F. Dellaert and M. Kaess, "Square root SAM: Simultaneous localization and mapping via square root information smoothing," *Int. J. Robot. Res.*, vol. 25, no. 12, pp. 1181–1203, 2006.

[10] M. Kontitsis, E. A. Theodorou, and E. Todorov, "Multi-robot active SLAM with relative entropy optimization," in *Proc. Amer. Control Conf. (ACC)*, Washington, DC, USA, 2013, pp. 2757–2764.

[11] H. Carrillo, Y. Latif, M. L. Rodriguez-Arevalo, J. Neira, and J. A. Castellanos, "On the monotonicity of optimality criteria during exploration in active SLAM," in *Proc. IEEE Int. Conf. Robot. Autom.*, Seattle, WA, USA, Jun. 2015, pp. 1476–1483.

[12] C. Stachniss, G. Grisetti, and W. Burgard, "Information gain-based exploration using Rao–Blackwellized particle filters," in *Proc. Int. Conf. Robot. Sci. Syst. (RSS)*, vol. 1. Cambridge, MA, USA, 2005, pp. 65–72.

[13] R. Sim and N. Roy, "Global a-optimal robot exploration in SLAM," in *Proc. IEEE Int. Conf. Robot. Autom.*, Barcelona, Spain, Apr. 2005, pp. 661–666.

[14] C. Stachniss, D. Hahnel, and W. Burgard, "Exploration with active loop-closing for FastSLAM," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, vol. 2. Sendai, Japan, 2004, pp. 1505–1510.

[15] K. Lenac, A. Kitanov, I. Maurović, M. Dakulović, and I. Petrović, "Fast active SLAM for accurate and complete coverage mapping of unknown environments," in *Proc. 13th Int. Conf. Intell. Auton. Syst.*, Padua, Italy, 2014, pp. 415–428.

[16] V.-C. Pham and J.-C. Juang, "An improved active SLAM algorithm for multi-robot exploration," in *Proc. SICE Annu. Conf.*, Tokyo, Japan, 2011, pp. 1660–1665.

[17] K. K. Leung and G. Gallagher, "Multi-robot localization and mapping strategy: Utilizing behavior based dynamic tree structure and observer-explorer routine," in *Proc. 3rd IEEE Int. Conf. Autom. Sci. Eng. (CASE)*, Scottsdale, AZ, USA, 2007, pp. 881–886.

[18] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, San Diego, CA, USA, 1994, pp. 3310–3317.

[19] D. Borrmann *et al.*, "A mobile robot based system for fully automated thermal 3D mapping," *Adv. Eng. Informat.*, vol. 28, no. 4, pp. 425–440, 2014.

[20] A. Kitanov and I. Petrović, "Exactly sparse delayed state filter based robust SLAM with stereo vision," in *Proc. 41st Int. Symp. 6th German Conf. Robot. (ISR) Robot. (ROBOTIK)*, Munich, Germany, 2010, pp. 1–7.

[21] M. R. Walter, R. M. Eustice, and J. J. Leonard, "Exactly sparse extended information filters for feature-based SLAM," *Int. J. Robot. Res.*, vol. 26, no. 4, pp. 335–359, 2007.

[22] M. Seder, M. Baotić, and I. Petrović, "Receding horizon control for convergent navigation of a differential drive mobile robot," *IEEE Trans. Control Syst. Technol.*, vol. 25, no. 2, pp. 653–660, Mar. 2017.

[23] S. M. LaValle, *Planning Algorithms*. New York, NY, USA: Cambridge Univ. Press, 2006.

[24] A. Konar, I. G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, "A deterministic improved Q-learning for path planning of a mobile robot," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 5, pp. 1141–1153, Sep. 2013.

[25] J.-C. Latombe, *Robot Motion Planning*. Dordrecht, The Netherlands: Kluwer, 1991.

[26] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA, USA: MIT Press, 2005.

[27] W. A. Kamal and R. Samar, "A mission planning approach for UAV applications," in *Proc. IEEE Conf. Decis. Control (CDC)*, Cancún, Mexico, 2008, pp. 3101–3106.

[28] J. Kleinberg and Éva Tardos, *Algorithm Design*. Boston, MA, USA: Pearson, 2006.

Ivan Maurović (M'11) received the B.Sc. and M.Sc. degrees in electrical engineering from the University of Zagreb, Zagreb, Croatia, in 2008 and 2010, respectively.

He joined the Department of Control and Computer Engineering, University of Zagreb, in 2010, as a Research Assistant of SEE-ERA.NET PLUS Project ThermalMapper, where he has been a Researcher of ACROSS Project, since 2012, and is currently a Researcher with the LAMOR Research Group. His current research interests include mobile robotics, exploration of unknown environmets, path planning, and localization.

Marija Seder (M'05) received the M.Sc. and Ph.D. degrees in electrical engineering from the University of Zagreb, Zagreb, Croatia, in 2004 and 2010, respectively.

She was a Visiting Researcher with the Autonomous Intelligent System Group, University of Freiburg, Freiburg im Breisgau, Germany, from 2012 to 2013, under the supervision of Prof. W. Burgard. She is currently an Assistant Professor with the Faculty of Electrical Engineering and Computing, Department of Control and Computer Engineering, University of Zagreb. Her current research interests include mobile robotics, especially motion planning, path planning, coverage planning, obstacle avoidance, and environment exploration.

Kruno Lenac (M'13) received the M.Sc. degree in electrical engineering from the University of Zagreb, Zagreb, Croatia, in 2013.

He is a Doctoral Research Fellow with the Faculty of Electrical Engineering and Computing, Department of Control and Computer Engineering, University of Zagreb. His current research interests include mobile robotics with a focus on robot localization and map building of unknown environments—SLAM, problem of connecting SLAM with path planning process, known as active SLAM.

Ivan Petrović (M'97) received the B.Sc., M.Sc., and Ph.D. degrees in electrical engineering from the University of Zagreb, Zagreb, Croatia, in 1983, 1989, and 1998, respectively.

He was an Research and Development Engineer with the Institute of Electrical Engineering, Končar Corporation, Zagreb, from 1985 to 1994. Since 1994, he has been with the Faculty of Electrical and Computing Engineering, University of Zagreb, where he is currently a Full Professor. He teaches a number of undergraduate and graduate courses in the field of control systems and mobile robotics. He has published over 40 journal and 160 conference papers, and results of his research have been implemented in several industrial products. His current research interests include various advanced control strategies and their applications to control of complex systems and mobile robots navigation.

Dr. Petrović is the Editor-in-Chief of the *Automatika Journal*. He is a member of the IFAC Technical Committee on Robotics, the FIRA Executive Committee, and the Croatian Academy of Engineering.