Hierarchical path planning of mobile robots in complex indoor environments

Marija Seder, Petar Mostarac and Ivan Petrović

Abstract

Inspired by the hierarchical D* (HD*) algorithm of D. Cagigas (Cagigas, 2005), in this paper we introduce a novel hierarchical path planning algorithm called focused hierarchical D* (FHD*). Unlike the original HD* algorithm, the FHD* algorithm guarantees the optimality of the global path, and it requires considerably less time for the path replanning operations. This is achieved with several modifications: (i) optimal placement of the so-called bridge nodes needed for hierarchy creation, (ii) focusing the search around the optimal path, which reduces the search area without loss of optimality, and (iii) introduction of partial starts and partial goals that further reduce computational time of replanning operations. The FHD* algorithm was tested in a multiroom indoor environment and compared to the original HD* algorithm, the nonhierarchical D* algorithm, and focused D* algorithm under the same conditions. The FHD* algorithm significantly outperforms other algorithms with respect to the computational time. Furthermore, it can be easily extended to the problem of path planning between different floors or buildings.

I. Introduction

An autonomous mobile robot is expected to provide flexible services in dynamic environments populated by other moving objects or/and animals and human beings. The major task that it should be capable to perform autonomously is finding and moving to the goal position given by the user or by the superimposed task planning and scheduling controller. The robot motion control system is a two-level system with path planning controller at higher level and path following controller at lower level. The task of the path planning algorithm is to compute the optimal path to the given goal and to replan the path in case the previously planned path is blocked by obstacles. The path following algorithm directly controls the robot motion with the aim to follow the planned path to the goal obeying robots kinematic and dynamic constraints. The path (re)planning in dynamic environments is in focus of this paper, where particular attention is paid to real-time issues of the planning process.

The majority of path planning algorithms produce a graph of possible paths to the goal (Latombe, 1991) and then an optimal path is found by a classical graph search algorithm such as A* algorithm (Nilsson, 1971), D* algorithm (Stentz, 1994) or focused D* (FD*) algorithm (Stentz, 1995). Two-dimensional (2D) occupancy grid maps are usually used to represent environments, where the grid is a rectangular array with equal spacing and the connectivity of the grid cells is considered to be eight-neighbor. Grid cells cover the area densely and each grid cell contains the information about the traversability and possibly uncertainty. As long as the spacing of the grid is sufficiently small, all the information needed for path planning has been preserved.

Classical graph search algorithms have two significant limitations. Firstly, resulting path is geometric curve with sharp edges as a multiple of 45 which is hard to follow due to kinematic and dynamic constraints of the mobile robot. This problem has been alleviated in various manners. For example, the Field D* algorithm extends standard D* algorithm by using linear interpolation to derive the path cost of points not sampled in the grid (Ferguson and Stentz, 2007). This algorithm efficiently produces very low-cost paths with a range of continuous headings, but in large environments is computationally inefficient. In our previous work (Seder and Petrović, 2007) the smooth trajectory is produced by integration of FD* algorithm and Dynamic Window local obstacle avoidance algorithm (Fox, Burgard and Thrun, 1997) considering explicitly mobile robot kinematics and dynamics and also ensuring collision-free motion among moving obstacles. Secondly, with a uniform resolution grid, these methods can be very memory and time intensive when the entire environment is represented at the highest resolution of the grid map. There are a number of approaches trying to solve this limitation. For example, quadtrees are used rather than uniform resolution grids in (Samet, 1982), but computed path can be suboptimal. Multi-resolution Field D* (Ferguson and Stentz, 2006) produces direct paths through a non-uniform resolution grid with lower computational time and memory.

In large complex indoor environments, such as multi-room floors, multi-floor buildings or group of buildings, the world model can become too large making path planning intractable or very inefficient. The plain graph information must be arranged to reduce complexity, gain efficiency and clarity. A suitable choice is hierarchical decomposition of the map based on hierarchical graphs (H-Graphs) (Fernández-Madrigal and González, 1998), (Fernández-Madrigal and González, 2002). Hierarchies of abstraction can reduce exponential complexity problems to linear ones (Korf, 1987). The interesting approaches to finding the problem of finding good abstraction hierarchies (Huang, Jing and Rundensteiner, 1997), (Giunchiglia, 1999), (Galindo, Fernández-Madrigal and González, 2004).

Hierarchical path planning is based on a refinement process through the hierarchy of abstractions and a reconstruction process that links partial paths obtained after refinement. In (Cagigas and Abascal, 2004) a set of optimal partial paths are

M. Seder, P. Mostarac and I. Petrović are with Faculty of Electrical Engineering and Computing, University of Zagreb, Unska 3, 10000 Zagreb, Croatia, e-mail: {marija.seder, petar.mostarac, ivan.petrovic}@fer.hr

previously stored in some nodes of the H-graph in order to speed up the refinement process. Even more strict demand on the computational time is required for path replanning in dynamic environments. The problem of many real-time algorithms is time or path cost nonoptimality. In (Cagigas, 2005) D* algorithm, which is widely used in dynamic environments, is converted into hierarchical D* (HD*) algorithm that uses precalculated path (materialization of costs). HD* algorithm improves computational time performance of the D* algorithm in large search spaces, including also three dimensional spaces, and therefore allows to obtain paths in a faster and more accurate way than traditional hierarchical path planning. Although HD* algorithm uses the set of precalculated partial optimal paths between key points, it is not ensured that the global path created as a union of these optimal partial paths is also optimal. Also, HD* algorithm uses heuristics towards the goal. However, heuristics towards the start would speed up the replanning process, (Stentz, 1995).

In this paper a new method is proposed that ensures path optimality and improves dynamic characteristics of the HD* algorithm. Path optimality is ensured by optimal placement of the key points between which the partial paths are calculated. Replanning process is accelerated by use of heuristics towards the introduced partial starts. Also, the partial goals are introduced from which the searching will continue in the replanning process. This procedure further shortens computational time.

The rest of the paper is organized as follows. In Section II a concept of the H-Graph is briefly described. In Section III classical search algorithms (A*, D*, FD*) and hierarchical search algorithm (HD*) are restated. Section IV describes proposed focused hierarchical D* algorithm (FHD*), which is an improvement of the HD* algorithm. In Section V experimental results obtained with FHD* are analyzed and compared to the results obtained with HD*, D* and FD* algorithms under the same circumstances. Finally, in Section VI some conclusions are pointed out.

II. Hierarchical map model

A hierarchical path planner is supported by a hierarchy of abstractions representing different views of a robot environment, i.e. 3D view, not physical but abstract. Thus, a graph-based hierarchical environment representation is needed.

A. H-Graph Definitions

The H-Graph is a sequence of hierarchical levels $L = L_0, L_1, ..., L_D$, where D is the depth of the hierarchy. L_0 is the "root level" which represents the most abstract description of an environment. L_D represents the most detailed description of an environment. Each level L_i, $(0 \leq i \leq D)$ contains also a graph $G_i = (N_i, A_i, C_i, W_i, T_i)$, where N_i is a set of nodes, A_i is a set of arcs, C_i is a set of Cartesian coordinates for N_i, W_i is a set of weights for A_i and T_i is a set of precalculated paths associated to N_i. The union of graphs G_0, G_1, G_2, ..., G_D is a graph $G = (N, A, C, W, T)$, where $N = N_0 \cup N_1 \cup ... \cup N_D$, $A = A_0 \cup A_1 \cup ... \cup A_D$, $C = C_0 \cup C_1 \cup ... \cup C_D$, $W = W_0 \cup W_1 \cup ... \cup W_D$, $T = T_0 \cup T_1 \cup ... \cup T_D$. An arc $a(n_J, n_K, w_H) \in A$ is defined by three elements n_J, n_K, w_H, where $n_J, n_K \in N$, $n_J \neq n_K$ and $w_H \in W$. A Cartesian coordinate $c_I \in C$ is defined by (x, y), where $x, y \in \mathbb{N}$. A weight $w_I \in W$ is real number $(w_I \in \mathbb{R})$.

B. Categories of nodes and associated functions

Nodes are classified into four classes: end, $cross$, $submap$ and $bridge\ nodes$. Submap nodes represent a subset of nodes in a deeper abstraction level of the hierarchy. The submap node set contained in N is called SN $(SN \subset N)$. The following functions are associated to submap nodes:

- map $\rightarrow n_J.map = n_K$, where $n_J \in L_j$, $n_K \in L_k$, $j = k + 1$ $(0 < j \leq D)$, $(0 \leq k < D)$ and $n_J \subset n_K$ in L_k. Map function shows in which node is n_J included in an upper level of the hierarchy. It means that n_K contains n_J in its submap.
- depth $\rightarrow n_J.depth = x$, where n_J is a node from L_x level. Function returns the level of the hierarchy which contains n_J.

End nodes are starting or goal nodes of a robot path planning. It means that they are included in L_D level of the hierarchy representing physical position of a robot in the environment. Cross nodes are subtargets which represent turnovers or crossings of the paths. Bridge nodes are nodes that connect submap to a parent submap and they are included in SN subset. The bridge node set included in N is called BN $(BN \subset N)$. The following functions are associated to bridge nodes:

- get_bridge_nodes $\rightarrow n_I.get\_bridge\_nodes = BN_I \subseteq BN$, where $n_I \in N$, $n_I.depth < D$ and BN_I satisfies $\forall n_x \in BN_I$, $n_x.map = n_I$. If n_I is a submap node, then the function returns the set of the bridge nodes which are included in n_I and are found in the next deeper level of the hierarchy.

Bridge nodes are divided into two classes: $horizontal\ bridge\ nodes$ and $vertical\ bridge\ nodes$. Horizontal bridge nodes follow the definition given earlier. Vertical bridge nodes are almost equal to horizontal bridge nodes but conceptually they connect two submaps that represent two floors in a building. Vertical bridge nodes are, therefore, connected to a physical infrastructure of a building. Usually elevator entrances are used as positions of vertical bridge nodes. An example of modeling a building as an H-Graph of floor levels with horizontal and vertical bridge node connections is shown in Fig. 1. Each floor level is represented by a submap node with noted other nodes in a deeper abstraction level of the hierarchy and a vertical bridge node that connects floor levels and serves for path planning between them.

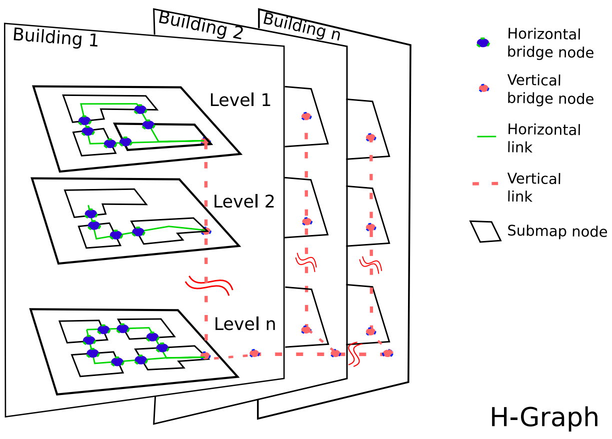

Fig. 1. Example of hierarchical H-Graph model of a building.

C. *Paths and associated functions*

Arcs (A) are non-directed which allows navigation in both directions between two nodes. Cartesian coordinates (C) are attributes associated to each node. They are used in the heuristic function of the path planning algorithm. Weights (W) indicate the cost of traversing arcs and are used by the cost function of the path planning algorithm.

A path is defined as a succession of nodes. The whole set of paths contained in an H-Graph is called P. For example, a path $P_I \in P$ of length L is defined by $P_I = (n_0, n_1, \dots, n_L)$, where $n_0, n_1, \dots, n_L \in N$ and $\exists\ a_0(n_0, n_1, w_{(0,1)})$, $a_1(n_1, n_2, w_{(1,2)}), \dots, a_{L-1}(n_{L-1}, n_L, w_{(L-1,L)}) \in A$. A path P_I has three attributes:

- cost $\rightarrow P_I.cost = x$, where $x \in \mathbb{R}$, assigns or gets path cost to P_I.
- length $\rightarrow P_I.length = L + 1$, where $L \in \mathbb{N}, P_I \in P$ and $P_I = (n_0, n_1, \dots, n_L)$, returns the path length of P_I.
- index $\rightarrow P_I.index(J) = n_J$, where $n_J \in P_I$, returns the node of a path in position J.

Each submap node $n_I \in N_i \subset N(0 \leq i \leq D)$ has its own precalculated path set $PPS_{nI} \in T_i \subset T(0 \leq i \leq D)$. The following functions are associated to a submap node n_I:

- pre_path $\rightarrow n_I.pre\_path(n_X, n_Y) = P_Z$, where $n_X, n_Y \in N$, $P_Z = (n_X, n_{X+1}, ..., n_Y)$, $P_Z \in PPS_{nI} \subset P$. If the requested path does not exist, the function returns NULL.

Precalculated paths are length optimal, calculated off-line between key points. Precalculated paths are grouped into three classes:

1) Paths which connect two bridge nodes within the submap node n_I.
2) Paths which connect the bridge nodes of n_I $(n_I.get\_bridge\_nodes)$ with the bridge nodes of its parent submap $((n_I.map).get\_bridge\_nodes)$.
3) Path which connects the closely related submap nodes, called "brother" submaps contained in n_I. Two submap nodes $n_X, n_Y \in N$ are "brother" submaps contained in n_I if $n_X.map = n_Y.map = n_I$, i.e. if they have the same "parent" submap in an upper level of the hierarchy.

With precalculated paths, recalculating several subpaths in a hierarchical search process is avoided. Thus, the extra storage space for the paths is required, but computational time is much lower because refinement of nodes in deeper levels of the hierarchy is avoided. Moreover, the H-Graph map model is very flexible and easily adapted. If the building map needs expanding, then the inner modularity of the H-Graphs solves the problem by adding precalculated partial paths between the new nodes and the old bridge nodes. In addition, the precalculated paths can be stored in a path set which can be added to or removed from the H-Graph.

D. *An example of the hierarchical map*

Following an example in (Cagigas and Abascal, 2004), one floor of a building is observed. The H-Graph has four abstract levels. At the most abstract level of the hierarchy (L_0), each floor is observed as a submap node, named Floor in Fig. 3. Level L_0 contains a graph G_0 composed of only floor nodes.

Level L_1 is composed of the floor sections and contains a graph G_1, which connects the nodes contained in the submap node $Floor$, Fig. 3. These nodes are named: *North*, *South*, *East* and *West*.

Each of these four nodes in the deeper level L_2 contains a graph G_{2i} ($\forall\ i \in North, South, East, West$), which connects the nodes contained in the corresponding submap. Floor section has its share of the belonging rooms. That introduces an extra information about the position of a room. Figs. 4 and 5 show the level L_2 and the corresponding graph G_{2North}.

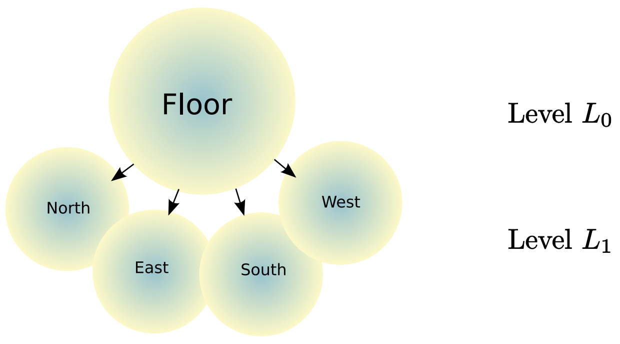

Fig. 2. First two most abstract levels of the hierarchy, L_0 and L_1.

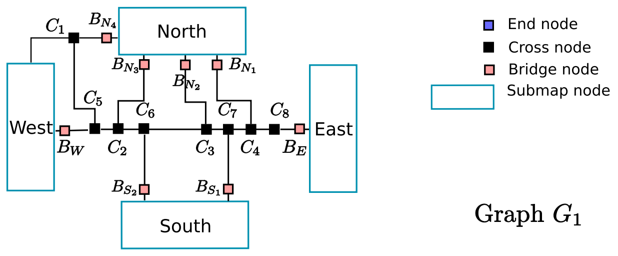

Fig. 3. Corresponding graph of the level L_1, graph G_1.

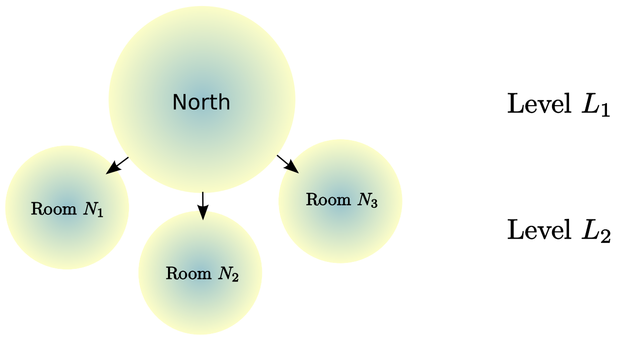

Fig. 4. The division of section $North$ to its corresponding rooms in the level L_2 of the hierarchy.

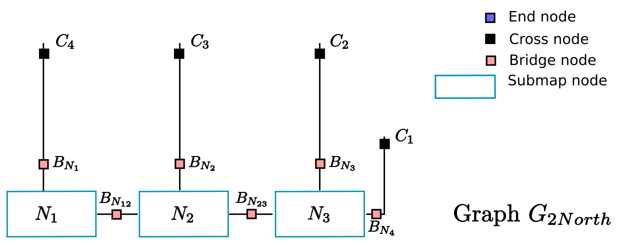

Fig. 5. Graph G_{2North} consists horizontal bridge nodes associated to cross nodes in graph G_1.

Each room in the deepest level L_3 contains a graph G_{3i} ($\forall\ i \in N_1, N_2, N_3$) which connects the nodes contained in the corresponding submap. Further division is not necessary. Figs 6 and 7 show the deepest level L_3 and the corresponding graph G_{3N1}. These nodes contain all the information needed for the path planning algorithm (real position, traversability, etc.) and allow positioning of a mobile robot at every available location within a room (i.e. end nodes). These nodes are on-line updated as the environment changes.

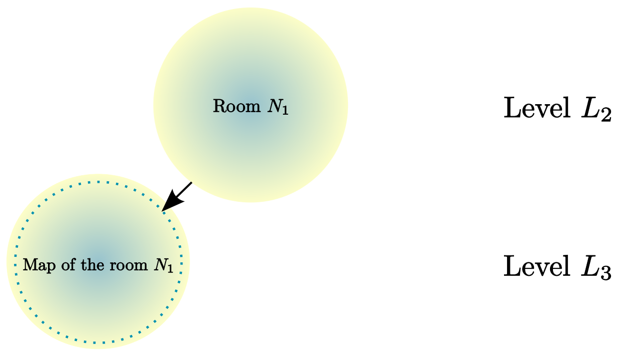

Fig. 6. Node N_1 in the level L_3 represents the map of a room.

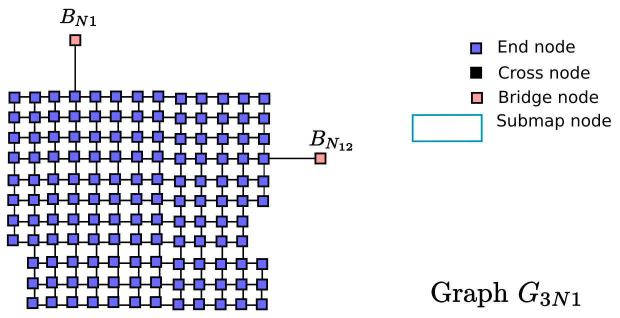

Fig. 7. Graph of the room N_1 which defines detailed description of internal space, its connection to other rooms in the same sector (section) and the connection to other sectors using the horizontal bridge nodes.

III. Path planning

A. Classical search algorithms

Graph based search algorithms are the most commonly used algorithms for path planning of mobile robots. Among them the most popular one is A* algorithm (Nilsson, 1971), which finds complete and optimal path in static environments. It is based on path cost functions g and h. Function g represents the total cost from the start node of the search to the current node. It is calculated as follows: $g = \sum w_i$, $i = 1, ..., d_N$, where d_N is the depth of the currently reached node in the search tree and w_i is the cost of traversing an arc when node ni is expanded with its neighbor node. Function h is a heuristic function, which estimates, but never overestimates, the cheapest path cost for achieving the goal node from the current node in the $(x,\ y)$ grid map search space. Such a heuristic function is called admissible and optimistic. The total cost function $f = g + h$ determines the order of expanding the nodes in the state space. When following any path from the start node the value of the f-function never decreases, which is true if heuristic exhibits monotonicity.

The path cost is based on some metric such as distance, time, energy expended, risk, etc. In our implementation, the path cost is based on distance travelled. The most often used heuristic function is the Euclidean distance from the current node to the goal node. However, the Euclidean distance is computationally inefficient since calculation of the square root function for each node expansion demands floating point arithmetic. In order to alleviate this problem a heuristic that uses integer arithmetic is proposed. This is possible in occupancy grid maps because they enable transition costs to be described as integer multiplies. For example, if each cell in a grid is regarded as a node in the graph, length of the edge e can be used for straight transition (e.g. $e = 10$cm) and length of the diagonal d for diagonal transition (e.g. $d = 14$cm). We used the following heuristic:

$$\begin{aligned} a &= \max(\ |x_G - x_N|, |y_G - y_N\ |\ ), \\ b &= \min(\ |x_G - x_N|, |y_G - y_N\ |\ ), \\ h(N) &= db + e(a - b), \end{aligned} \tag{1}$$

where (x_N, y_N) are the coordinates of the current node N and (x_G, y_G) are the coordinates of the goal node G. This heuristic exhibits monotonicity because it fulfills the triangular inequality property.

A* search fans out from the start node, expanding neighbor nodes within the contours of increasing f-value until the goal node is reached. During the expansion, backpointers to the parent nodes are set. The A* uses backpointers to represent paths to the start.

In a dynamic environment the global path must be completely replanned each time the environment changes or a mobile robot follows the path imperfectly. The A* algorithm performs poorly since it does not use search information from previous iterations. Minimum path criterion may not be optimal in the sense of minimum time. This problem has been solved by D* graph search algorithm introduced in (Stentz, 1994), which allows updating of only those nodes along the path that actually change due to sensor measurements. The D* algorithm is similar to A* in the case of initial off-line path planning. The main distinction to the A* algorithm is that the D* algorithm search fans out not from the start node but from the goal node. Cost function g here represents the total path cost from the goal node to the current node. It is computed from the goal node since the search starts from the goal. Nodes are expanded within the contours of increasing g-value until the start node is reached. During the expansion, backpointers from each node in the searched area to its parent nodes are set. The D* uses backpointers to represent paths to the goal. Optimal paths to the goal from every node in the expanded area of the environment are computed simply by following the backpointers. Further on-line execution of the algorithm relies on sensor information about the robot environment. Any discrepancy that is discovered from the earlier information about the environment initiates algorithm execution. Nodes which traversal cost has changed are expanded first. Important functions for the replanning process are function g and so-called key function k, which is defined as

$$k(N) = \min(k(N), g_{new}), \tag{2}$$

where g_{new} is a new value of the cost function g of the expanded node N due to cost changes in the environment in the replanning process. Initially, k is equal to g. D* handles any path cost optimization problem where the cost parameters change during the search process by processing raise and lower states. Raise state is classified by relation $k < g$. It is used to propagate information about path cost increases. Lower state is classified by relation $k = g$. It is used to propagate information about path cost reductions. Order of expanding the nodes is defined by minimal value of the function k, where the cost change is propagated through the number of nodes around the optimal path. New path, if exists, is determined by redirecting the pointers locally. The replanning process stops when the cost of the best node examined is greater than the cost of the node of robot current position.

Including the heuristic in the remaining cost of the path to the start node leads to the focused D* algorithm (FD*) variant (Stentz, 1995). Heuristic is chosen similar to (1) but coordinates of the start node $S(x_S, y_S)$ are used instead of the goal node $G(x_G, y_G)$. The initial planning is similar as with the A* algorithm, but starts from the goal node. Replanning is similar as with the D* algorithm except the cost function f is now sum of the heuristic function h and the key function k ($f = k + h$).- Order of expanding the nodes is defined by minimal value of the cost function.

B. A.Cagigas hierarchical search algorithm

The key part of Cagigas HD* algorithm is the set of the precalculated paths. Paths between bridge nodes are calculated by A* algorithm (heuristic function is calculated towards the goal). A partial path defines connection between two bridge nodes with a certain cost. If a path between two bridge nodes exists then these bridge nodes are considered as neighbor nodes in a search process. When calculating the initial path, (Cagigas and Abascal, 2004), the start node and the goal node are treated as bridge nodes and partial paths are calculated between them and their adjacent bridge nodes. There is no search process at the deepest level of the hierarchy and their corresponding graphs. Instead, the search algorithm tries to find the best path between the submaps where the start and the goal nodes are included. These submaps are linked to their parent submaps in an upper level of the hierarchy through their bridge nodes. Process ends when the set of partial paths that joins the start and the goal node is constructed. It is a typical bottom-up process used when working with hierarchical graphs.

When a broken arc is detected in the initial path in the mobile robot vicinity, the path should be replanned. In (Cagigas, 2005) the path is replanned from the start node, which represents the current robot position, to the goal node. The heuristic function (1) is calculated towards the goal node and order of expanding the nodes depends on the total cost function $f = g + h$, as in A* algorithm. The cost function g represents the total cost from the start node to the current node of the search. Planning ends when a node included in the initial path is found. When the first bridge node from the deepest level is reached, hierarchical widening begins. Because every bridge node at the deepest level contains partial paths towards other bridge nodes from the upper level, planning is shifted upwards in the hierarchy. In upper levels, submap nodes are expanded according to function $f = g + h$ until the goal submap node is reached. Here, function g is calculated as a sum of the materialized costs of the precalculated paths between bridge nodes, $g = \sum P_i.cost$, $i = 1, ..., d_N$, where d_N is the depth of the currently reached bridge node in the search tree at the upper level and $P_i.cost$ is actually the weight, i.e. the cost of traversing an arc between start and goal bridge node of the partial path P_i. Thereby, at higher levels only bridge nodes exists with their arcs and weights. Function h is again calculated according to (1). Since precalculated paths may include submap nodes too, these nodes must be substituted by associated precalculated path for the final solution (node unrolling).

In (Cagigas and Abascal, 2004) and (Cagigas, 2005), placement of bridge nodes is not considered, although it is crucial for the performances of the hierarchical planning. If the bridge nodes are placed at arbitrary positions in the environment, they may be placed apart from the optimal path between the start node and the goal node. In that case, a union of partial optimal paths between the bridge nodes may not be optimal.

IV. Focused hierarchical D* search algorithm

With the goal to ensure optimality of the global path and to decrease computational complexity of path replanning in real-time, we propose three important modifications of the HD* algorithm: (i) optimal placement of the bridge nodes, (ii) focusing the search around the optimal path, and (iii) introduction of partial starts and goals. We named our algorithm as the focused HD* algorithm (FHD*).

A. Optimal placement of the bridge nodes

Lets assume that an indoor environment can be geometrically divided into rooms and halls. In L_{D-1} level there are submap nodes each representing a room with related bridge nodes that connect it with other rooms. Links between nodes respect geometrical structure of the environment. Vertical bridge nodes are placed at positions of stairs and elevators, i.e. exactly at physical connections of a floor/building with its neighboring floors/buildings. They are the local goals necessary for taking the next level. If there is a room with more than two bridge nodes it will certainly contain a cross node, since more than one path through it exists. Here, cross nodes are not of importance for path planning since placement of bridge nodes keeps path optimality. L_D level for the i-th room is associated to respective graph G_{Di}, which is a part of the occupancy grid map and contains geometrical characteristics of the environment. Each cell in the grid map corresponds to a node in G_{Di}. It would be perceived that flexibility of hierarchical map that is adapted here to physical structure of the environment is used. If there are many rooms at level L_{D-1} they can be grouped together in a submap, which represents a sector of the environment by adding next level, L_{D-2}. Also, by adding other floors to L_0, the number of nodes at the level would grow, while by adding one whole new building to the map one more abstract level would be added and it would contain only two nodes representing those two mentioned buildings (Cagigas and Abascal, 2004), (Cagigas, 2005).

Bridge nodes that connect submap nodes must be placed in the optimal path that should be followed by a mobile robot on its way from one room to the other. When a mobile robot follows the optimal path, the trajectory safeness is of great importance. In a narrow passage the safest path would be one that goes through the middle of the passage. Since D* and FD* algorithms are capable of handling continuous values for arc costs, a potential field (Khatib, 1986) can be constructed by assigning the highest cost values to the occupied cells and decreasing the cost values for unoccupied cells proportionally to their distance from the obstacles. Obviously, in narrow passages middle cells will have minimal cost values, and therefore the optimal path would pass through them. Therefore, it is logical to place the bridge nodes in the middle of the doors separating the rooms, as shown in Fig. 8.

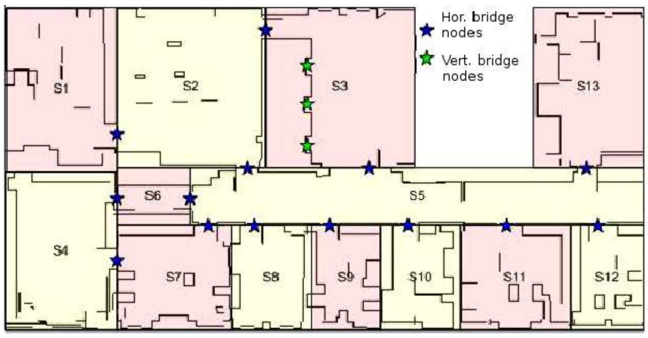

Fig. 8. Map of the floor level with division on submap nodes and positions of bridge nodes.

In our implementation of the potential field integer values of costs are used in the form of so-called safety cost mask (Seder, Maček and Petrović, 2005). The safety cost mask is included at the deepest level L_D, associated to the occupancy grid map. The grid map is composed of unoccupied cells and occupied cells. Each cell in the grid map within the safety cost mask around an occupied cell receives a corresponding safety cost value, which depends on its distance from the occupied cell. The utmost cells of the safety cost mask obtain a cost value for one greater than unoccupied cells out of the safety cost mask, and safety cost value of inner cells incrementally increase from utmost cells to the occupied cell. The size of the cost mask can be determined as in our previous work, (Seder and Petrović, 2007). Therefore, the optimal path would go through the middle of the narrow passages, i.e. exactly where bridge node is placed.

Fig. 9 represents the detailed description of a part of a hall. Real coordinates of the grid map cells are noted on the $x-$ and $y-$axes. The cells containing obstacles are C-space enlarged prior to an execution of the path search algorithm to account for the robot dimensions. Cost values of the enlarged obstacle cells are presented with black colour and of cells free of obstacles with white colour. Shades of grey correspond to incrementally higher cost value closer to the obstacles. Obstacle cells of the real obstacle placement (without accounting for the robot dimensions) are marked with x-marks. The cells in the grid map have assigned unique identifiers used for spatial indexing purpose. Cell S_{ij} is uniquely defined with

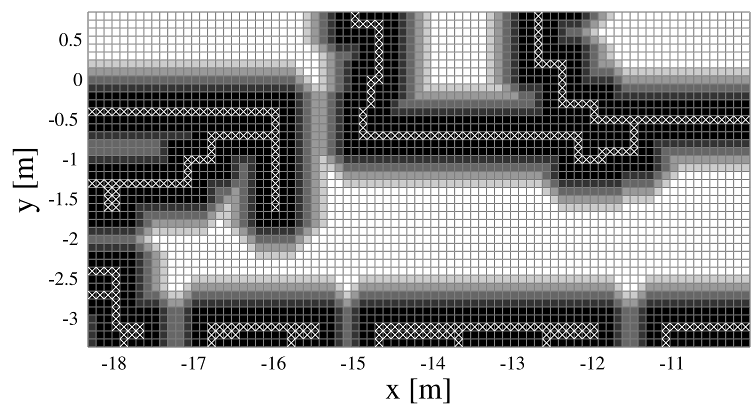

Fig. 9. The occupancy grid map with cost mask at level L_D.

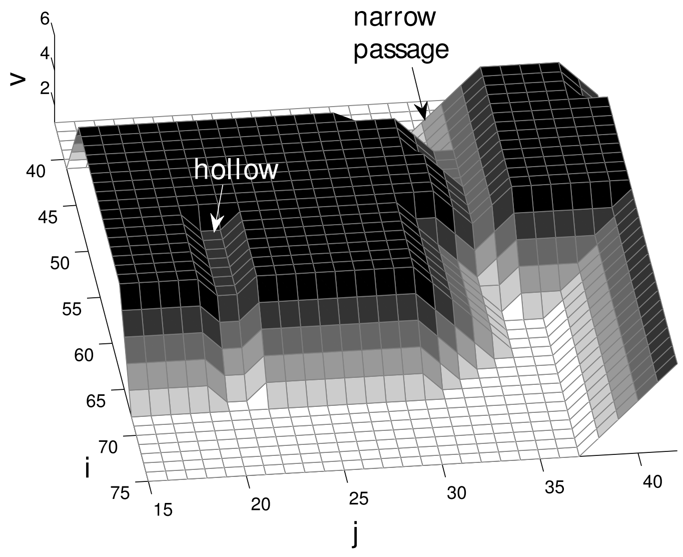

Fig. 10. Three dimensional view of the cost grid map: function v.

grid map coordinates (i, j) given by $(\lfloor \frac{x}{d_{CELL}} \rfloor, \lfloor \frac{y}{d_{CELL}} \rfloor)$, where x, y are real coordinates of the middle point of the cell in the environment and d_{CELL} is the edge length of the cell. Occupancy grid map with the cost mask is considered as a function of two variables $v : (i, j) \mapsto V$, where domain is the Cartesian product of two sets of grid map coordinates, $i \in I = \{0, 1, 2, ..., M_I\}$ and $j \in J = \{0, 1, 2, ..., M_J\}$, where M_I, M_J are defined by the size of the environment, and codomain V is a set of cost values assigned to each cell in the grid map. Minima of the function v are at cells out of the safety cost mask and maxima are at occupied cells. Local minima and local maxima along one coordinate are significant for narrow passages detections. The cell at which local minimum is present, but is not also the global minimum, could be the place of a narrow passage or it could be some hollow in the obstacle. The local maximum of the other coordinate at the same cell would confirm that it is really a narrow passage, as is illustrated in Figure 10. Finding of narrow passages is actually equivalent to finding of saddle points of the function v as follows:

$$\begin{array}{l} v_i^{''}(i,j) = v(i-1,j) - 2v(i,j) + v(i+1,j), \\ v_j^{''}(i,j) = v(i,j-1) - 2v(i,j) + v(i,j+1), \\ \forall n_{ij} \in G_D : \text{sign}(v_i^{''}(i,j)) = -\text{sign}(v_j^{''}(i,j)), \end{array} \tag{3}$$

where n_{ij} is a node from the graph G_D associated to cell S_{ij}, $i \in I$ and $j \in J$, whose coordinates fullfills given expression. More details can be found in (Seder, Jurić-Kavelj and Petrović, 2008).

In (Cagigas and Abascal, 2004) it is pointed out that each submap is connected to a bridge node in the next deeper level of the hierarchy. Since we placed bridge nodes at the doors each of them is shared by two neighbor rooms. Therefore, number of bridge nodes is almost halved comparing to the original HD* algorithm, what additionally accelerates execution of the FHD* algorithm.

If there is a path between two submap nodes it has to contain a bridge node. Optimality in sense of the path cost relies in the fact that once assembled path certainly contains partially optimal paths. A classical search algorithm takes care of the optimality of the path between two bridge nodes. In Fig. 11 it is shown that the optimal path produced by the D* algorithm between the start node and the goal node, which are not in the same submap at some level of the hierarchy, is exactly the same as the path composed of precalculated paths produced by the D* algorithm. Length of the path as a number of cells between two nodes $N_1, N_2 \subset N$ is noted as $l_{(N_1,N_2)}$. It can be seen that the path from the start node to the goal node in the upper subfigure, Fig. 11, is of the same length as the sum of the partial path lengths in the lower subfigure. It should be also noticed that nodes BN_1 and BN_2 are counted twice. Since D* algorithm does not use heuristic, it is not optimal in the computational time and memory usage, so usage of the FD* algorithm is preferred. The FD* algorithm uses heuristic in order to minimize computational time for the optimal path finding. If the free space in the environment is wide enough to allow different paths with the same cost, path computed by the FD* algorithm may differ geometrically from the path computed by the D* algorithm, as shown in Fig. 12. Nevertheless, the path computed by the FD* algorithm is optimal by the cost and geometrical differences are not of any importance. Comparing the corresponding paths in Fig. 12 and Fig. 11, it can be observed that lengths of both paths are equal.

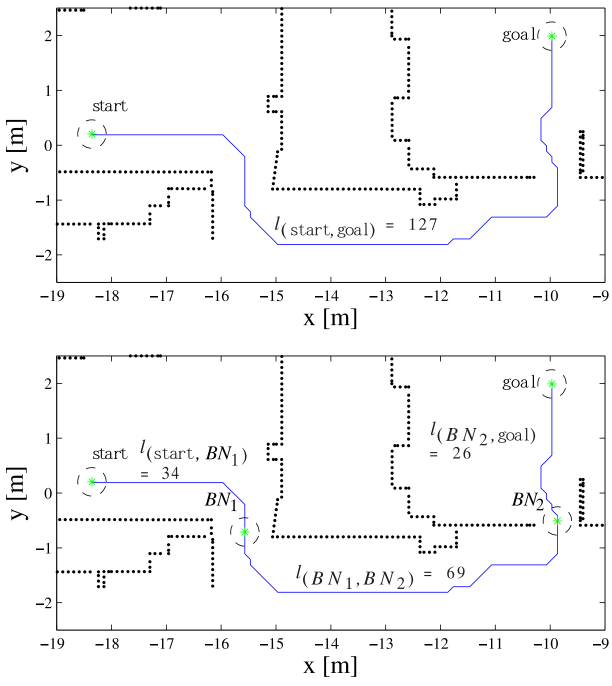

Fig. 11. Optimal path obtained by D* (upper subfigure) and union of partial paths obtained by D* (lower subfigure).

B. *Focusing the search around the optimal path*

In (Cagigas, 2005) heuristic towards the goal is used for expanding the best nodes according to the function $f = g + h$. The cost that is estimated by the heuristic function can be replaced by the exact cost from the goal node to the current node. A mobile robot on its way to the goal continuously changes its position. When a new unknown obstacle blokes the optimal path, replanning process uses robot current position as the start node. Therefore, it will be more useful to calculate heuristics toward the moving start position, since its cost continuously changes, whilst the goal is at fixed position. Replanning by the A* algorithm is slower than by the FD* algorithm. In case of the A* algorithm there is no direct path to the goal from all of the nodes in the searched area, as is the case with the FD* algorithm. That information in the FD* algorithm is simply reused for fast replanning. On the contrary, in the A* algorithm nodes are pointing towards start node and that information is not useful for replanning. Also, because of the heuristic property of nonoverestimating the path cost, for the A* algorithm it follows:

$$f(start) \leq f(n) \leq f(goal), \tag{4}$$

while for the FD* algorithm the relation is opposite:

$$f(goal) \leq f(n) \leq f(start), \tag{5}$$

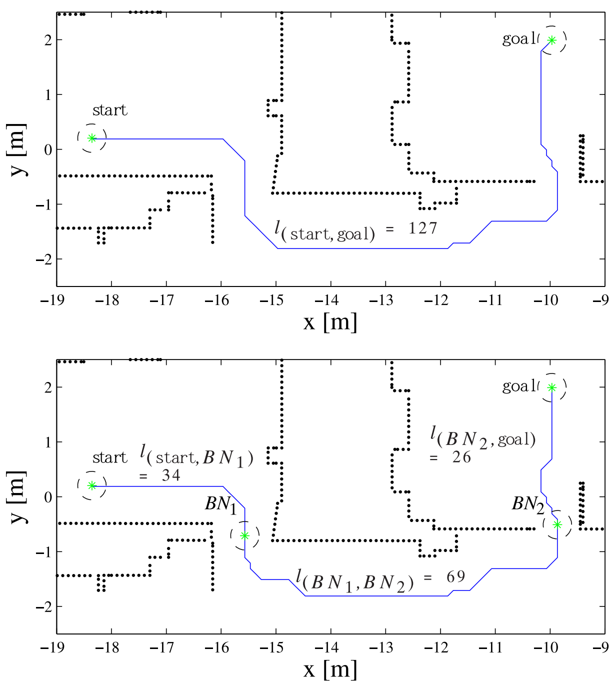

Fig. 12. Optimal path obtained by FD* (upper subfigure) and union of partial paths obtained by FD* (lower subfigure).

where n is a node in the optimal path between the start node and the goal node and f is a total cost function defined as $f = g + h$ for the A* algorithm and $f = k + h$ for the FD* algorithm. Since the nodes are explored orderly according to the minimal value of the function f, the FD* algorithm chooses nodes closer to the goal firstly, while the A* algorithm chooses nodes closer to the start firstly. Therefore, replanning by the A* algorithm starts with the start node, while replanning by the FD* algorithm starts with the newly occupied node (the obstacle node). Redirecting the pointers locally around the obstacle node is much faster than exploring nodes from the start node.

Since a set of precalculated paths between bridge nodes is constant and unchangeable for a map, it can be calculated off-line. The FHD* algorithm calculates initial path similarly as the HD* algorithm. The start node and the goal node are treated as bridge nodes and partial paths are calculated between them and their adjacent bridge nodes by the FD* algorithm. In upper levels, bridge nodes are expanded according to the function $f = g + h$ until the start node is reached. Here, function g is calculated from the goal node to the current bridge node as a sum of the materialized costs of the precalculated paths between bridge nodes, $\sum P_i.cost$, $i = 1, ..., d_N$. Function h is heuristic towards the start node computed as with the FD* algorithm. During the expansion, backpointers to the parent nodes are set. These backpointers are used as in the FD* algorithm, to represent paths to the goal node. Optimal paths to the goal from every bridge node are computed simply by following the backpointers.

Also, every expanded node has assigned calculated cost values of function g and k (defined as in the FD* algorithm), which is important for further path replanning process. Accordingly, additional attributes are added to the path P_I:

- g $\rightarrow P_I.g(J) = x$, where $x \in \mathbb{R}$, assigns or gets path cost g to $P_I.index(J)$ (the node of a path in position J).
- k $\rightarrow P_I.k(J) = x$, where $x \in \mathbb{R}$, assigns or gets path cost k to $P_I.index(J)$.

C. Partial starts and partial goals

Partial starts and partial goals are introduced in the replanning process of the FHD* algorithm. Their purpose is lowered computing time and memory usage, without loss of optimality. If a mobile robot detects a new unknown obstacle blocking the initial path, replanning process is engaged. It is divided into three steps:

1) Replanning at level L_D: A newly occupied node that first occurs in the initial path followed by a mobile robot is noted as N_o. Replanning is done only in the submap containing N_o. This submap is called the obstacle submap. In the case that N_o is exactly the bridge node, the submap that is next to the bridge node in the initial path is considered as the obstacle submap. Node that contains N_o is expanded first. Nodes that point to N_o have their cost enlarged (raise states). This propagation of cost enlargement spreads backwards to the partial start node (N_{ps}), which is defined as follows. The submap that contains robots current position (N_c) is called the current submap. If N_c is the bridge node, the current submap is considered as the submap next to N_c in the initial path. Partial start node N_{ps} is N_c if the current submap and the obstacle submap are the same submap. Otherwise, N_{ps} is the bridge node that connects these two submaps. A special case is $N_o \subset BN$, where N_{ps} is the first free node before N_o in the initial path. Fig. 13 illustrates the example of this three cases of obstacle placement. In the left subfigure N_c and N_o are in the same submap $Room_i$. N_{ps} is then N_c. In the middle

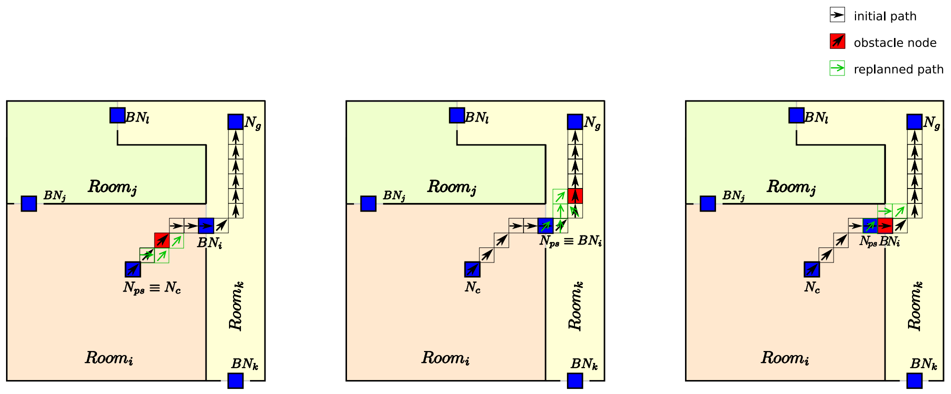

Fig. 13. Three situations of placement of obstacle and current position. Path replanning at level L_D: obstacle in the current submap (left); obstacle in the neighbor submap (middle); obstacle at the bridge node (right).

subfigure N_c is in submap $Room_i$ and N_o is in submap $Room_k$. Partial start N_{ps} is then BN_i. In the right subfigure N_o is BN_i. N_{ps} is one node before BN_i. Order of expanding the nodes is defined by minimal value of function $f = k + h$, where k is defined as in FD* and h is heuristic towards N_{ps}. When node which will lower the cost of the previously raised states is expanded, redirecting the pointers will form a new path around the obstacle, as illustrated in Fig. 13. Replanning process at level L_D stops when there are no relevant nodes with the cost less or equal than the cost of N_{ps} in the obstacle submap. If there is no path in the obstacle submap, i.e. the path is completely closed with an obstacle, N_{ps} will have prohibitively large cost. With an updated cost of N_{ps}, searching is continued upwards in the hierarchy. An example of possible three situations, where no path is found in the obstacle submap, is shown in Fig. 14. In the left subfigure an obstacle is placed in

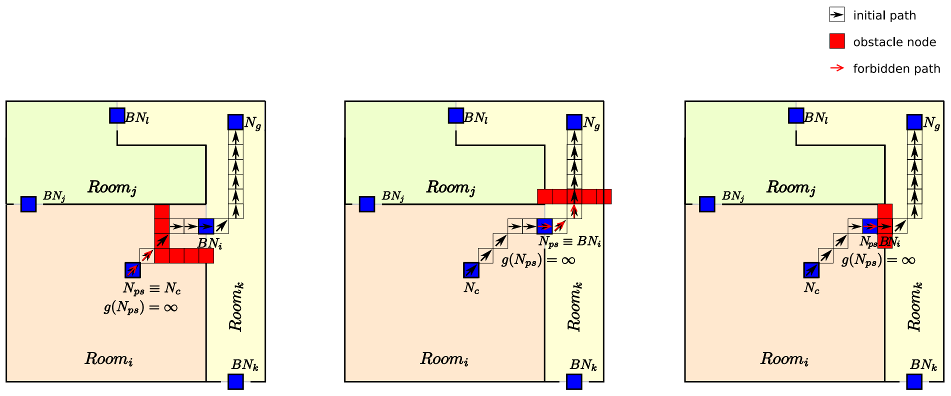

Fig. 14. Three situations of placement of obstacle and current position. No path is found at level L_D: obstacle in the current submap (left); obstacle in the neighbor submap (middle); obstacle at the bridge node (right).

the current submap. There is no path from N_{ps} to BN_i in the obstacle submap $Room_i$. In the middle subfigure an obstacle is not placed in the current submap. There is no path from N_{ps} to N_g in the obstacle submap $Room_k$. In the right subfigure N_o is BN_i and there is no path from N_{ps} to N_g in the obstacle submap $Room_k$.

2) Replanning in upper levels of the hierarchy: Updated node N_{ps} is expanded first. Expanding of nodes is done similarly as in level L_D, but hierarchically. That means that bottom-up process is used and all bridge nodes that point to N_{ps} in upper levels raise their state. Here, heuristic towards N_c is calculated and nodes are expanded according to minimal values of function $f = k + h$. All nodes that could not be expanded due to unknown path cost are remembered for the final step of replanning. Those nodes do not have a partial path with N_c at level L_D. Therefore, optimal path could not be yet completed. When all sufficient bridge nodes have updated their costs, searching is shifted again at the deepest level of the hierarchy. Fig. 15 illustrates a configuration of nodes at level L_{D-1} that corresponds to the configuration of nodes in Fig. 14, the right subfigure. All costs to the goal node that have been changed (function g) due to the replanning process are noted. Remaining step is linking N_c with this solution.

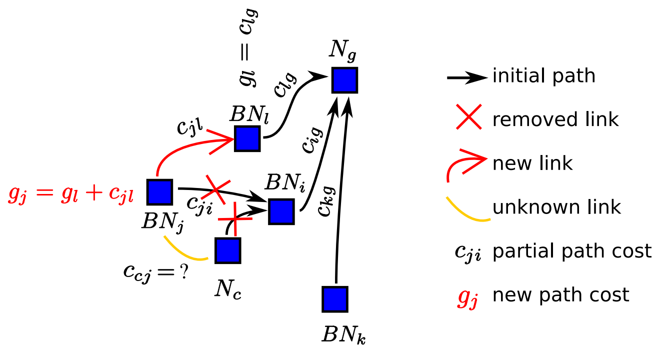

Fig. 15. Replanning of bridge nodes at level L_{D-1}.

3) Linking process at level L_D: Every bridge node, which does not have partial path with N_c in the current submap is examined. Partial paths should be calculated to ensure optimality of the total path. If more than one path cost should be calculated, D* algorithm is used in the most effective way. Instead of calculating separately each path cost, all path costs are determined with one initial search execution by D* algorithm. Robot current position N_c is the partial goal N_{pg} for exploring the whole current submap by D* algorithm. Nodes are expanded according to minimal values of function $f = g$, until all sufficient bridge nodes are reached, or all possible nodes in the current submap are exhausted. From every free node in the searched area optimal path and path cost to N_{pg} are determined. If only one path cost must be calculated, instead of D*, FD* should be used in order to minimize number of searched nodes. Fig. 16 shows the remaining partial path calculated from N_{pg} to BN_j. This remaining partial path is linked with initially calculated partial paths between nodes BN_j, BN_l

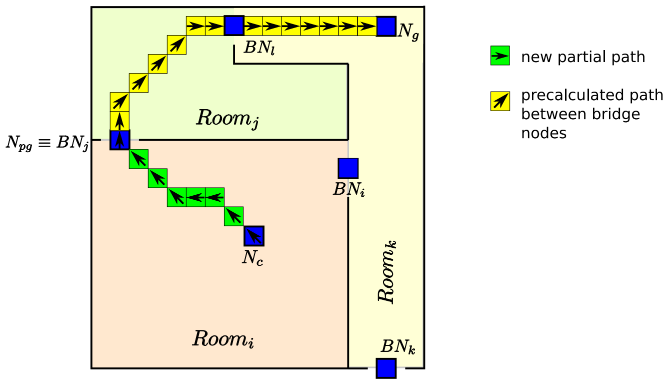

Fig. 16. Linking process at level L_D.

and N_g (the goal node).

V. Test results

The proposed path planning algorithm has been implemented in *Player/Stage* (a free software tool for robot and sensor applications, `www.playerstage.sourceforge.net`) and tested on Pioneer 3DX mobile robot (manufactured by Mobile Robots Inc.) at our Department. We compare FHD* algorithm with D*, FD* and HD* algorithms under the same circumstances. A 2D occupancy grid map of the Department is built and organized into hierarchical levels. The deepest level L_D contains nodes that are matched to the occupancy grid map which describes details of the mobile robot environment. Mobile robot is equipped with laser range finder that is used to detect dynamic obstacles and to update occupancy grid map information.

In L_{D-1} level there are nodes that represent individual rooms and halls. Fig. 17 shows grouping of nodes from level L_D into rooms. Each room is presented by a different color. Bridge nodes are noted with black $\diamond$-marks. For clarity, only first four rooms and bridge nodes are named. In L_{D-2} level there are nodes that represent floors. Algorithms are tested in one floor of a building and, therefore, L_{D-2} level contains only one node. The algorithm described in (Seder and Petrović, 2007) is used for path following.

In Fig. 18 the set of precalculated paths for the optimal path generation between the start node (in $Room_1$) and the goal node (in $Room_4$) is shown. This set is used by both hierarchical algorithms, HD* and FHD*. So, advantage of placing the bridge nodes at optimal places is not pointed out and is considered that HD* is working with the same bridge nodes as FHD*. Therefore, initial searching is the same for both algorithms.

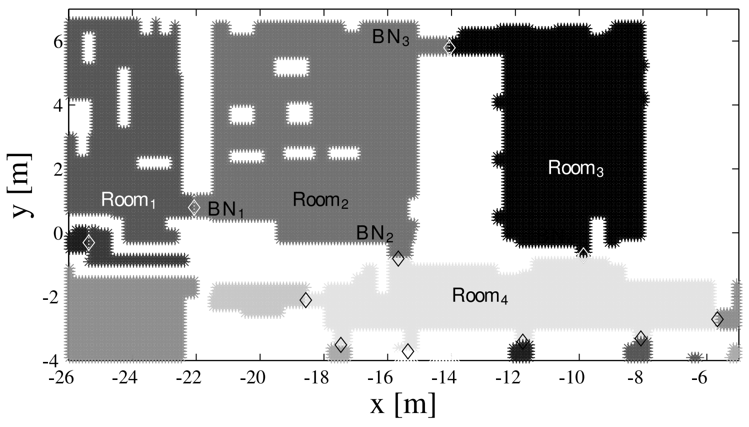

Fig. 17. Submap nodes in level L_{D-1} representing rooms and bridge nodes between rooms.

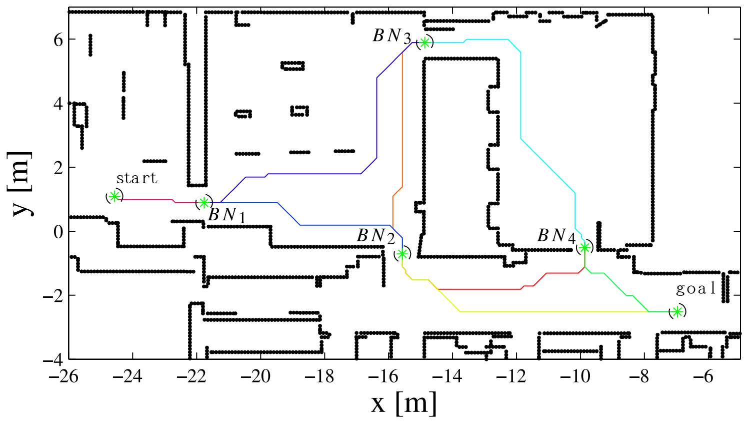

Fig. 18. The set of precalculated paths between bridge nodes.

The experiments are organized in a way that once robot detects an unknown obstacle in the initial path, path must be replanned at the higher level of the hierarchy. In Fig. 19 initial and replanned paths are compared for FHD* and FD* algorithms. All paths are optimal, but geometrically different because of already mentioned characteristic of using heuristics

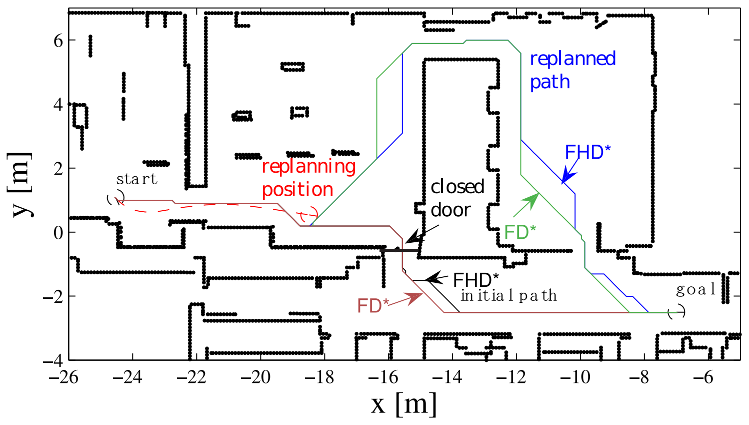

Fig. 19. The comparison of the initial and replanned paths computed by FHD* and FD*.

towards different starts. When robot traverses to position noted as “replanning position” in Fig. 19 it detects by laser range finder that the door is closed. Traversed trajectory is noted with dashed line. The path comparison for HD* and D* is not necessary since they also produce optimal and similar paths.

Next figures show searched area for FD*, D*, HD* and FHD* algorithms. They are purposely ordered according to the number of expanded states. In Fig. 20 searched area by FD* algorithm is shown. Inner area is searched initially and outher

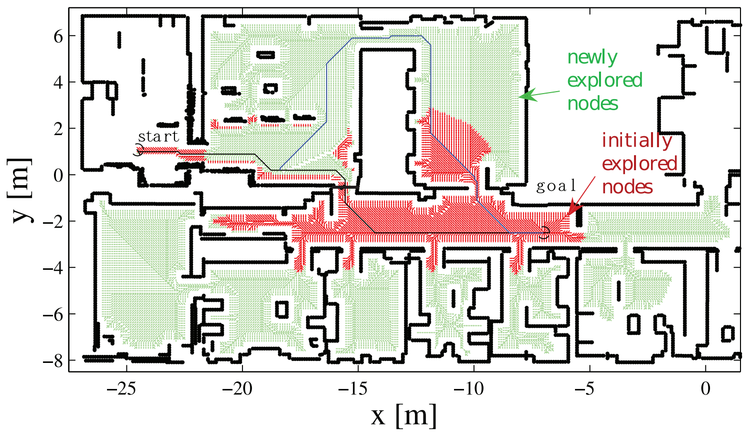

Fig. 20. The searched area by FD* algorithm.

when new obstacle appears. Since FD* uses heuristics, minimal area is searched initially. Therefore, replanning process continues where initial planning stops and further extends the area until the new path is calculated. In Fig. 21 searched area by D* algorithm is shown. Since D* does not use heuristics, the area that spreads radially around the goal is searched

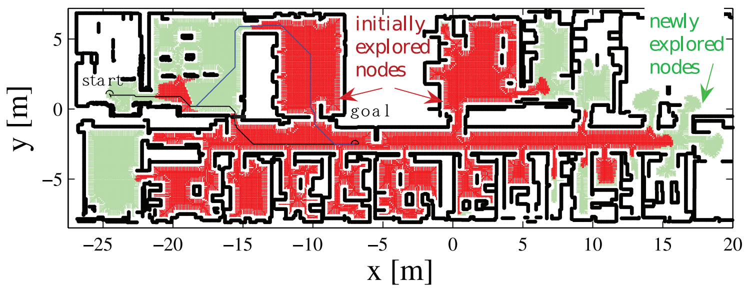

Fig. 21. The searched area by D* algorithm.

initially. Therefore, replanning area is a little bit smaller than in the case of FD* algorithm, since the path from every node in the area is computed initially. In Fig. 22 searched area by HD* algorithm is shown. The area searched initially is based

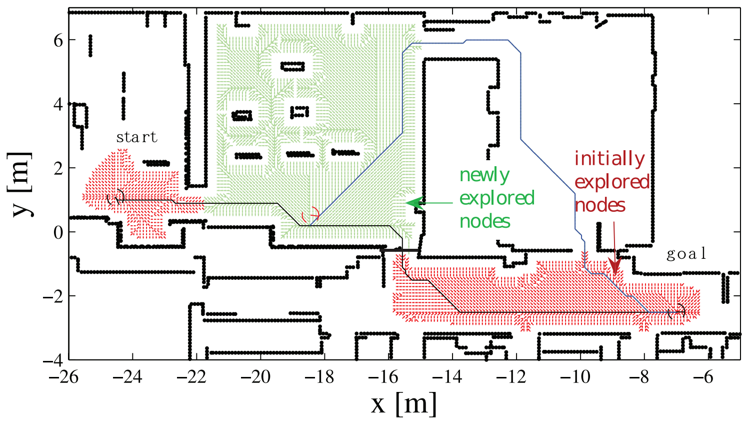

Fig. 22. The searched area by HD* algorithm.

on the distance of the start/goal node from all the bridge nodes in its submap that are linked to bridge nodes of the goal/start submap. There is only one path computed from the start node to the bridge node in the start submap and corresponding searched area is small. There are two paths computed from the goal to the two bridge nodes in the goal submap that lead to the start submap. In the replanning process, besides a couple of bridge nodes updated at the higher level of the hierarchy, only one whole room is searched which saves computational time significantly. In Fig. 23 searched area by FHD* algorithm is shown. Initially searched area is the same as in HD*, but, the replanning search is further decreased. Results of the

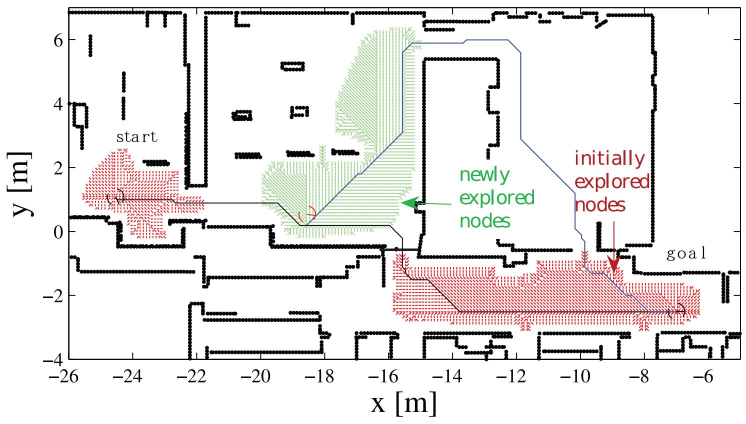

Fig. 23. The searched area by FHD* algorithm.

algorithm comparison from the computational point of view are presented in Tab. I for initial calculations and in Tab. II for replanning calculations. Since the path cost is tightly associated with the path length, it is equal for each algorithm.

TABLE I
COMPARISON OF PATH PLANNING ALGORITHMS IN INITIAL PROCESS

Alg.	path length [cells]	num. of explored nodes	num. of iterations	planning time [ms]
D*	187	27360	21063	272 ms
FD*	187	6767	4783	36 ms
HD*	187	3150	2082	25 ms
FHD*	187	3159	2082	25 ms

TABLE II
COMPARISON OF PATH PLANNING ALGORITHMS IN REPLANNING PROCESS

Alg.	path length [cells]	num. of explored nodes	num. of iterations	replanning time [ms]
D*	188	14518	11395	120 ms
FD*	188	17796	12358	130 ms
HD*	188	5822	5571	70 ms
FHD*	188	3398	2945	41 ms

Number of explored nodes is the key indicator of the computational complexity of the algorithms. Number of iterations is associated to re-doings of the main *while* loop of the search algorithm.

VI. CONCLUSION

In this paper the FHD* algorithm is proposed as extension of the HD* algorithm of Cagigas. Several important modifications were made in order to improve algorithm behavior during on-line path planning (i.e., path replanning). In particular, the path cost, computational time, and memory usage are significantly reduced. We proposed the strategy for the optimal placement of bridge nodes a key component when creating the set of precalculated (i.e., partial) paths thus guaranteeing the optimality of the total path. By using heuristics towards robots' current position the replanning process is focused around the optimal path. As a consequence, the new optimal path is found by examining minimal number of nodes. Partial goals and partial starts are introduced for fast replanning and creation of new links between hierarchical levels. The D*, FD*, HD* and FHD* algorithms were experimentally tested and compared under the same conditions. The experiments confirmed expected advantages of hierarchical planning algorithms (HD* and FHD*). The set of precalculated paths ensures fast replanning in the case of changing the goal position, given either by the robot user or superimposed task planning and scheduling algorithm. Furthermore, since replanning by the FHD* algorithm is very fast, one could also expect fast response in environment populated with moving obstacles.

VII. ACKNOWLEDGMENTS

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

Cagigas, D. 2005. Hierarchical D* algorithm with materialization of costs for robot path planning. *Robotics and Autonomous Systems* **52**(2-3), 190–208.

Cagigas, D. and Abascal, J. 2004. Hierarchical Path Search with Partial Materialization of Costs for a Smart Wheelchair. *Journal of Intelligent and Robotic Systems* **39**(4), 409–431.

Ferguson, D. and Stentz, A. 2006. Multi-resolution Field D. *Intelligent Autonomous Systems 9: IAS-9* .

Ferguson, D. and Stentz, A. 2007. 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.

Fernández-Madrigal, J.-A. and González, J. 1998. Hierarchical graph search for mobile robot path planning. *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on* **1**, 656–661.

Fernández-Madrigal, J.-A. and González, J. 2002. Multihierarchical graph search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**(1), 103–113.

Fox, D., Burgard, W. and Thrun, S. 1997. The dynamic window approach to collision avoidance. *Robotics & Automation Magazine, IEEE* **4**(1), 23–33.

Galindo, C., Fernández-Madrigal, J.-A. and González, J. 2004. Improving efficiency in mobile robot task planning through world abstraction. *Robotics, IEEE Transactions on [see also Robotics and Automation, IEEE Transactions on]* **20**(4), 677–690.

Giunchiglia, F. 1999. Using Abstrips Abstractions–Where do We Stand?. *Artificial Intelligence Review* **13**(3), 201–213.

Huang, Y., Jing, N. and Rundensteiner, E. 1997. A Hierarchical Path View Model for Path Finding in Intelligent Transportation Systems. *GeoInformatica* **1**(2), 125–159.

Khatib, O. 1986. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *The International Journal of Robotics Research* **5**(1), 90–98.

Korf, R. 1987. Planning as search: a quantitative approach. *Artificial Intelligence* **33**(1), 65–68.

Latombe, J. 1991. *Robot Motion Planning*. Kluwer Academic Publishers. Dodrecht, Netherlands.

Nilsson, N. 1971. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill Pub. Co.

Samet, H. 1982. Neighbor Finding Techniques for Images Represented by Quadtrees.. *Computer Graphics and Image Processing* **18**, 37–57.

Seder, M., Jurić-Kavelj, S. and Petrović, I. 2008. Automatic Creation of Hierarchical Maps for Indoor Environments. *Proceedings of the Fifth International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS 2008)* pp. 19–24.

Seder, M., Maček, K. and Petrović, I. 2005. 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* pp. 1785–1790.

Seder, M. and Petrović, I. 2007. Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. *Robotics and Automation, 2007 IEEE International Conference on* pp. 1986–1991.

Stentz, A. 1994. Optimal and efficient path planning for partially-known environments. *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on* pp. 3310–3317.

Stentz, A. 1995. The focussed D* algorithm for real-time replanning. *Proceedings of the International Joint Conference on Artificial Intelligence* **8**.