Preliminary version in Proc. of 7th RoboCup Int. Symposium, Padua, Italy, 2003.
In B. Browning, D. Polani, A. Bonarini, and K. Yoshida (editors): RoboCup-2003:
Robot Soccer World Cup VII, LNCS 3020, pp. 332-343, Springer, 2004.

Local Multiresolution Path Planning*

Sven Behnke

International Computer Science Institute
1947 Center St., Berkeley, CA, 94704, USA
behnke@icsi.berkeley.edu

Abstract. Grid-based methods for finding cost optimal robot paths around obstacles are popular because of their flexibility and simple implementation. However, their computational complexity becomes unfeasible for real-time path planning if the resolution of the grid is high.
These methods assume complete knowledge about the world, but in dynamic environments where sensing is done on board the robot, less is known about far-away obstacles than about the ones in close proximity. The paper proposes to utilize this observation by employing a grid of variable resolution. The resolution is high next to the robot and becomes lower with increasing distance. This results in huge savings in computational costs while the initial parts of the paths are still planned with high accuracy. The same principle is applied to the time-axis, allowing for planning paths around moving obstacles with only a moderate increase in computational costs.

1 Introduction

Path planning is an important subtask of the robot navigation problem, which is to find a path from a start configuration to a target state and to traverse it without collision. The navigation problem can be decomposed into three subtasks: mapping and modeling the environment, path planning, and path traversal with collision avoidance.

Many approaches to path planning have been described in the literature [8, 9]. They can be grouped into local and global methods. Local path planning methods do not attempt to solve the problem in its full generality, but use only the information available at the moving robot to determine the next motion command. One well known local path planning technique is the potential field method [7]. Here, the robot follows the gradient of a force field. The field is generated by attractive potentials pointing towards a target and by repulsive potentials that point away from obstacles. The potential field method has a low computational load and generates smooth paths that stay away from obstacles. However, the greedy gradient descent may get trapped in local minima. It is hence most useful in environments where local minima are unlikely. Furthermore, it can be used for fast reactive obstacle avoidance.

* This work was supported by a fellowship within the postdoc program of the German Academic Exchange Service (DAAD).

In contrast, global methods assume complete knowledge about the world. They frequently rely on the concept of free space, the configurations the robot can take without collision [10]. It is convenient to shrink the robot to a point while growing the obstacles accordingly to obtain the free space.

Roadmap path planning methods inscribe a graph into the free space that contains all possible paths. For instance, a roadmap defined by a visibility graph [11] can be used to find the shortest path around polygonal obstacles. Another possibility to define a roadmap is to use Voronoi borders [12] as graph edges in order to find a path that stays far away from obstacles. To rapidly explore high-dimensional configuration spaces planners have been proposed that randomly sample configurations and connect samples in free space by simple local paths, thereby creating probabilistic roadmaps [6]. One disadvantage of these methods is that only a binary representation (occupied/free) of the configuration space is possible.

Another class of global planning methods decompose the free space into cells. Exact cell decomposition results in cells of different simple shapes as required by the shape of obstacles. Approximate cell decomposition methods use predetermined cell shapes, sizes, and positions to approximate the free space [1]. Popular approximate cell decompositions include uniform coverage with square cells and quadtree representations [5] that use smaller cells next to the obstacle borders.

Once the decomposition is determined, dynamic programming can be used to find an optimal path. This requires to fill out a data structure, e.g. a multidimensional table, that contains solutions for all possible subproblems. If the resolution of a decomposition is high or the state space has many dimensions this can still be computationally demanding.

The computational efficiency of path planning is essential for online-problems [2], where paths are planned and executed in real time, for on board planning, where the computational resources are limited, and for planning in dynamic environments, where frequent replanning is required. All three of the above constraints are present in many leagues of the RoboCup competition.

On the other hand, in dynamic environments a detailed path from the start to the target has little chance of execution. Obstacles move unexpectedly as the robot traverses the path and hence continuous replanning is required. Furthermore, due to limited local sensing capabilities, far-away obstacles can be determined only with reduced accuracy. These observations motivate the local multiresolution path planning method proposed in this paper. The idea is to use high resolution to represent the configuration space in close proximity to the robot and to lower the resolution with increasing distance from the robot. This concentrates the planning resources to the begin of the path, the part that must be traversed first and where the information about obstacles is most reliable. While the computational load is reduced dramatically, the immediate movement of the robot can still be planned with high accuracy.

The paper is organized as follows. The next section describes grid-based path planning and details its extension to the multiresolutional case. In Section 3, the traversal of planned paths and the effects of the initial robot motion are covered.

```
PlanPath(target, obstacles, 𝒩){
    grid = eval = ClearGrid();
    q = InitPriorityQueue();
    while (!q.empty()){
        p = q.pop();
        if (target == p)
            return previous;
        for(n∈ 𝒩(p)){
            if(eval(n) == 0)
                best = ∞;
            else
                best = eval(n);
            new = p.cost + grid(p)·n.l0 + getGrid(n, obstacles, grid)·n.l1;
            if(new<best){
                eval(n) = new;
                q.push([n, new, getHeuristic(n, target)]);
                previous(n) = p;
            }
        }
    }
}
```

Fig. 1. A^* search for the least cost path to the target. The cost of a configuration is computed by getGrid and stored in the grid array. A heuristic function is used to guide the search. The cost of the shortest path to a configuration is maintained in the eval array. The reverse path is produced by traversing the previous list starting at target.

Section 4 applies the multiresolution idea to the time axis to find paths around moving obstacles. The paper concludes with a discussion of the experimental results and indicates possibilities for future work.

2 Grid Based Path Planning

Grid-based path planning methods decompose the configuration space into an array of cells. Costs are associated with the cells to represent the occupancy by obstacles. Neighboring cells are connected by edges. The cost of an edge can be derived from the cost of the two cells it connects. An minimal cost path can be found by searching this graph.

2.1 Basic Method

The basic algorithm used to find the least cost path in such a graph is summarized in Figure 1. It maintains in the eval array the cost of the best path seen so far to each of the configurations reached. The costs of grid cells are computed by getGrid as new cells are explored and stored in the grid array for future use. The edges of the graph are given as adjacency list $\mathcal{N}$ for each grid point. Each directed edge contains two lengths, l_0 and l_1, that describe the distance to the cell border from its start and its end node, respectively. The cost of an edge is

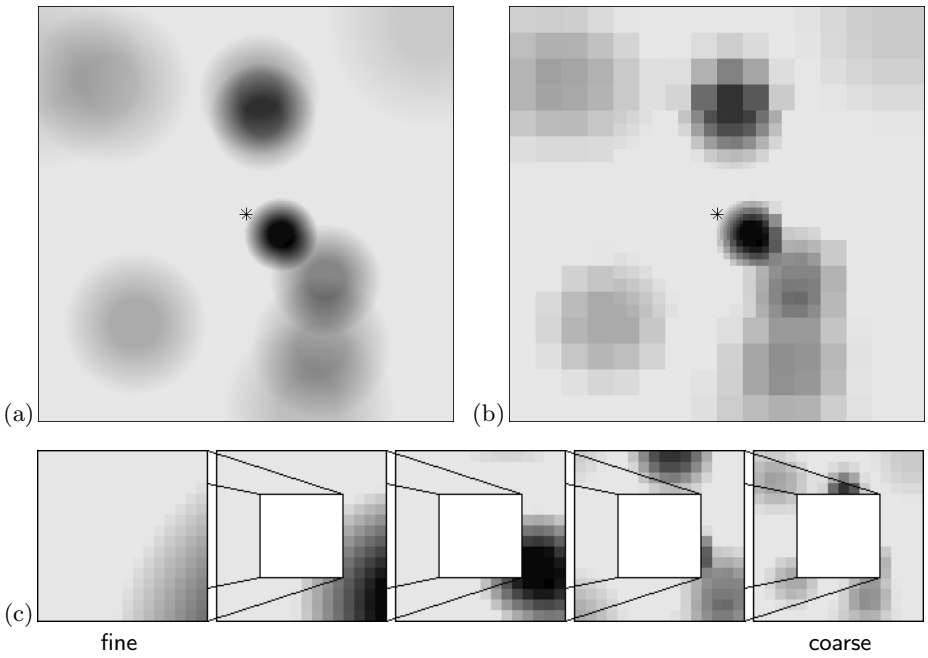

Fig. 2. Cost function used for the experiments. The robot is located in the center of the grid, as indicated by the star. Obstacles are represented by fuzzy discs. With distance from the center, the discs become larger and their height (indicated by darkness) decreases. (a) single resolution 256×256; (b) multiple resolutions 16×16×5 shown overlaid; (c) multiple resolutions 16×16×5 shown side by side.

computed as weighted sum of both grid values. The cost of a path is the sum of its edges.

InitPriorityQueue initializes the search with the start nodes. Since the algorithm is used here for local search around the current robot position, the search always starts at the center of the grid. The algorithm expands the nodes first that have the lowest accumulated cost until the best path cannot be improved any more.

2.2 Cost Function

The cost function that describes the occupancy of a grid cell can be chosen arbitrarily. Here, simple disk-like obstacles are modeled, as illustrated in Figure 2(a). 10 obstacles are placed at random positions. Their radius $r = r_o + r_r + r_d$ consists of a fixed component, which represents the radius of the obstacle r_o plus the radius of the robot r_r, and a variable component r_d that increases linearly with distance from the grid center. The far-away obstacles are modeled larger, because their position can be sensed with less accuracy from the perspective of the robot and because they might move before the robot gets close to them.

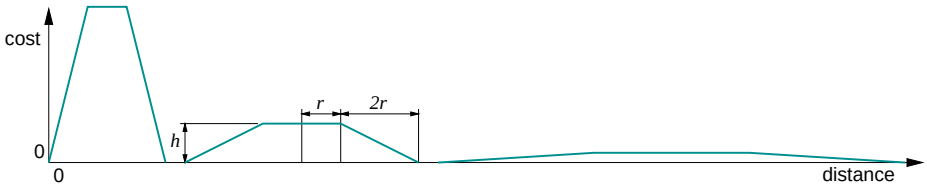

Fig. 3. Cut through cost function for obstacles with different distances from the robot.

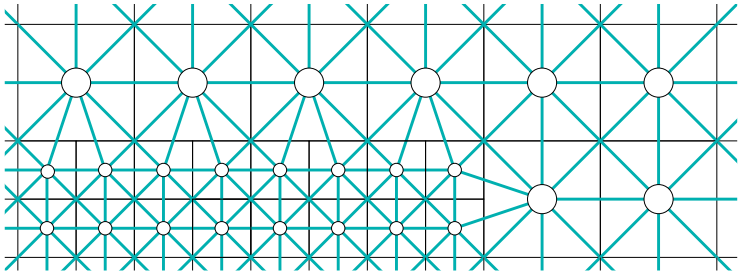

Fig. 4. Connectivity of the multiresolutional cell grid. A detail of the border between two resolution levels is shown.

Each obstacle is also characterized by a height h which is inversely proportional to the squared radius to keep its integral constant. The cost increase of a grid cell that is caused by an obstacle depends on their distance, as illustrated in Figure 3. It is constant if the distance is smaller than the radius and decreases linearly to zero at three times the radius. To compute the cost of a grid cell, the contributions from all obstacles are added to a uniform base cost.

The cost function is a simple and flexible way to express uncertainty. Obstacles with noncircular shapes could be included into the cost function in an analogous way.

2.3 Non-Uniform Resolution

It is not necessary to represent the entire grid with a high uniform resolution. Since far-away obstacles cover a larger area, a coarser resolution suffices there to approximate them. This is illustrated in Figure 2(b). Here, the resolution is high in the center of the grid and decreases towards the outside. This corresponds to the situation shown in Figure 2(c). Multiple low-resolution grids of size $M \times M$ are stacked concentrically. The inner part $([\frac{1}{4}M \ldots \frac{3}{4}M][\frac{1}{4}M \ldots \frac{3}{4}M])$ of a grid is not used, but the next grid level is placed there, until the highest resolution is reached. To cover the same area as a uniform $N \times N$ grid with the same inner resolution, only $K = \log_2(N/M) + 1$ levels of size $M \times M$ are needed. If N is large compared to M this lowers the number of grid cells substantially. In the following experiments, I use $N = 256$, $M = 16$, and $K = 5$. Hence, the flat grid has 64 times as many cells as the multiresolutional grid.

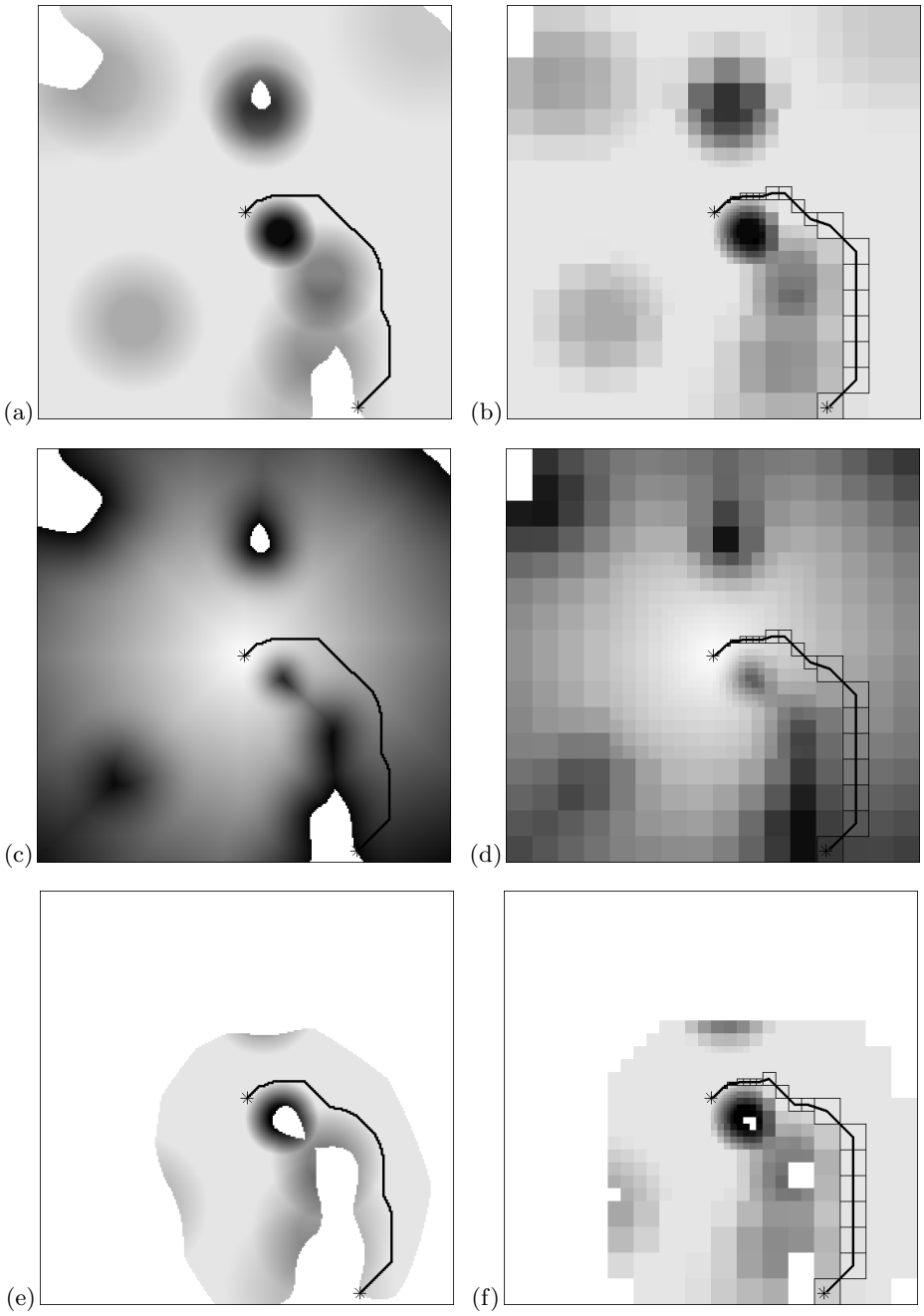

Fig. 5. Path planning without (a-d) and with (e-f) heuristics using a flat (a,c,e) and a multiresolutional (b,d,f) grid. (a,b,e,f) show the status of the grid and (c,d) show the status of the of the eval array after the search terminated at the target (lower star).

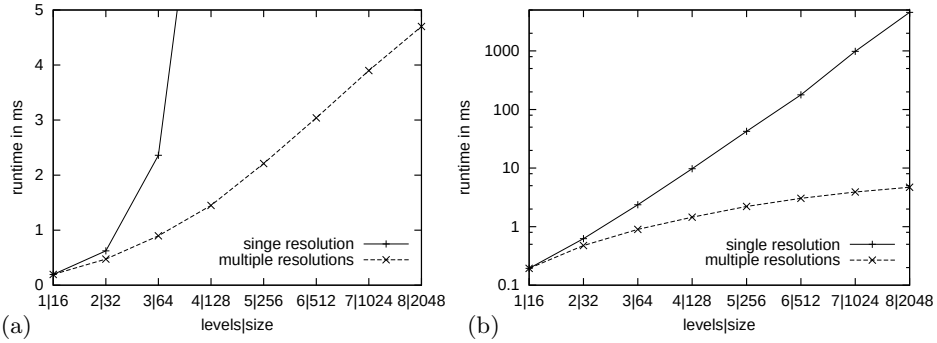

Fig. 6. Runtime of the flat and the multiresolutional search for different problem sizes: (a) linear scale (the multiresolutional search grows approximately linear in K); (b) logarithmic scale (the flat search grows approximately exponential in K).

The connectivity within cells of the same level of this multiresolutional hierarchy is set to the 8-neighborhood. Care must be taken at the borders between resolution levels to connect the neighboring cells. Figure 4 illustrates the connectivity that is used for the experiments. Except for the corners, each high-resolution cell connects to two adjacent low-resolution cells and each low-resolution cell connects to four high-resolution cells.

2.4 Heuristics

The A^* algorithm [4] is an efficient and well studied best-first search algorithm. It uses a heuristic function to guide the search. This function is an optimistic estimate of a path's total cost.

Since each grid cell has at least the base cost, the remaining part of the path from a grid point to the target cannot be less expansive than the Euclidean target distance weighted by the base cost. Hence, the sum of the accumulated cost of the best path to a grid point plus this heuristics can be used to determine the expansion order. As can be seen in Figure 5, the use of this heuristics can substantially lower the number of visited grid cells. The altered expansion order may alter the path found only if two paths have the same costs.

The figure also compares the algorithm for the flat and the multiresolutional grid representation. One can observe that the produced paths are very similar. In particular, the start of the multiresolutional path is as detailed as the path produced using the flat grid.

2.5 Runtime

The different cell numbers between the flat and the multiresolutional grid result in different runtimes. Figure 6 displays how this difference grows with the problem size. The runtimes represent the measured average running time of the path

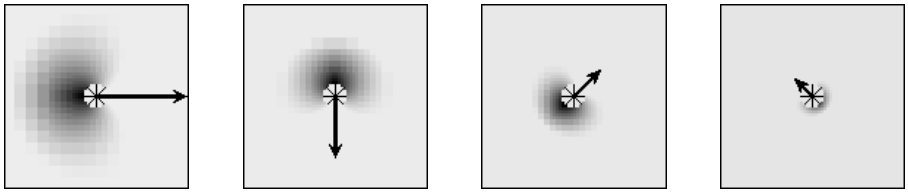

Fig. 7. Obstacle placed behind a moving robot to account for its initial velocity. It discourages sudden changes in direction.

planner to random targets with random obstacles. A 1 GHz Athlon processor has been used for the measurement. The algorithm has been implemented in C++. At the leftmost data point, where $K = 1$ and $N = M = 16$, both representations are identical. As N gets larger, K is adjusted accordingly. One can observe that the runtime grows approximately exponential with K when a single resolution is used and grows, after some cache effects, approximately linear when multiple resolutions are used. This corresponds well to the growth of the cell numbers. For $K = 8$ and $N = 2,048$, the flat search needs on average 4.55s while the multiresolutional search needs only 4.70ms on average.

3 Continuous Planning and Execution

In a dynamic environment, a planned path cannot simply be executed. Since the obstacles move, the plan must be updated as the robot follows its trajectory. Furthermore, in order to make consecutive plans compatible, the initial robot motion must be taken into account.

3.1 Initial Condition

One simple way to account for the initial velocity of the robot is to place an additional obstacle behind it, as shown in Figure 7. This obstacle favors paths that initially continue in a similar direction the robot is already moving. The larger the robot's initial velocity, the more severe a sudden change in direction would be and hence the more pronounced this obstacle must be.

3.2 Partial Execution and Replanning

Figure 8 illustrates how two different initial conditions lead to two different paths. The figure also shows, how the robot generates a trajectory by moving along the initial segment of the path. The path is continuously updated. As the robot comes closer to initially far-away obstacles, their radius decreases, since their position can now be determined with greater precision and they are less likely to move. Hence, the robot passes these obstacles closer than originally planned.

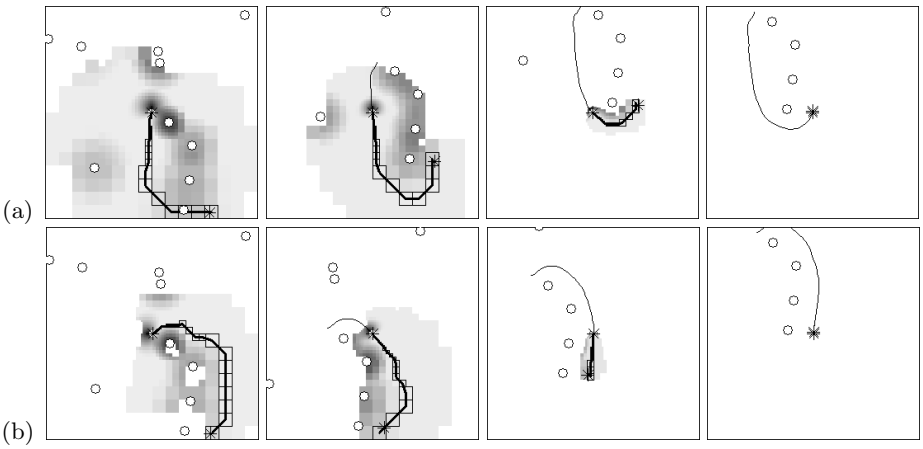

Fig. 8. Different initial conditions lead to different paths. As the path is executed, the robot is followed by the obstacle representing its velocity. (a) initial downward movement; (b) initial rightward movement. The thin line indicates the generated trajectory.

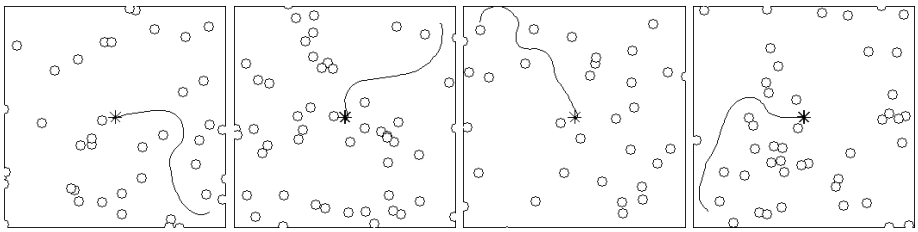

Fig. 9. Paths executed in an environment with many obstacles. The robot started in one of the corners and generated the trajectory while driving to the target (star).

Figure 9 shows some additional trajectories that have been generated in an environment with more obstacles. The trajectories are smooth, relatively short, and stay away from obstacles. Hence, they are suitable to reach the target fast while avoiding the chance of collisions.

4 Dynamic Planning

If the movement of obstacles can be estimated, a dynamic path can be planned by extending the dimensionality of the configuration space [3]. Figure 10 shows how the time axis can be represented in a multiresolutional fashion. For illustrative purposes the robot's position has been reduced to a single dimension.

Since time advances only in one direction, the higher-resolution arrays are not centered in the middle of the time-axis, but are located at its start. Hence, the first time-steps of the path are modeled with high precision while later time-steps are longer. This leads only to a moderate increase in computational complexity.

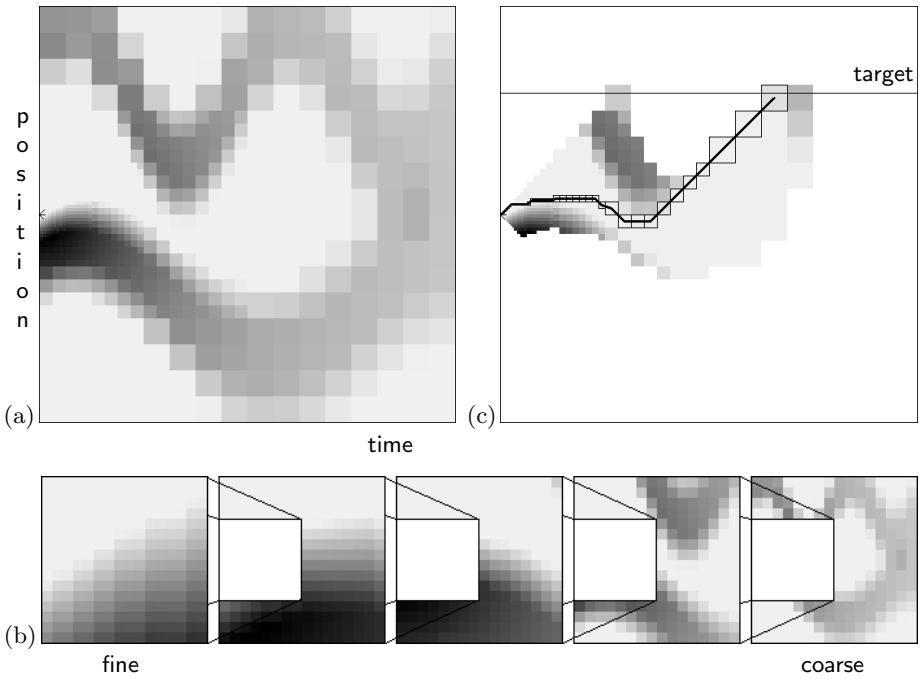

Fig. 10. Dynamic planning. The horizontal axis corresponds to time while the vertical axis represents space. The robot is located at the center of the left edge. (a) multiresolutional cost function 16×16 ×5 shown overlaid; (b) cost function 16×16 ×5 shown side by side; (c) planned path to the target position, represented by a horizontal line.

As can be seen, obstacles are not circular any more, but look like a line that becomes wider and flatter with distance from the origin. The two obstacles shown move along sinusoidal trajectories. To plan a minimal time path to a target-position not a single target cell, but a line of cells at this position and all points of time must be considered as search target.

Part (c) of the figure shows a planned path. One can see that the robot first moves upwards to avoid the lower obstacle, then moves downwards to avoid the upper obstacle and finally moves straight to the target. A direct motion to the target is not possible, since the maximum speed of the robot has been set to one. This is reflected in the connectivity shown in Figure 11. The edges are a subset of the edges from Fig. 4. Only edges that advance in time and do not exceed the maximum speed are included.

5 Conclusions

The paper proposed a local multiresolutional path planning algorithm. In contrast to quadtree algorithms [5] that focus the computational resources at the obstacle borders, this algorithm represents the configuration space next to the

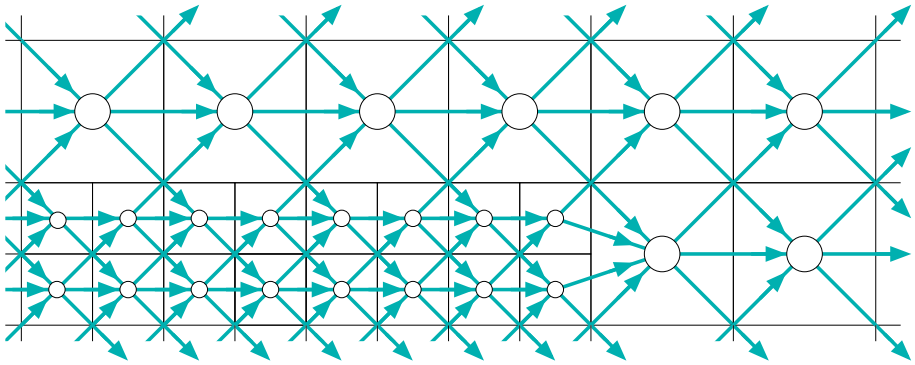

Fig. 11. Edges for dynamic planning. A detail of the border between two resolution levels is shown. All edges advance in time. The maximal speed is one.

robot with higher resolution than far-away from it. This leads to the use of fewer grid cells, as compared to a representation that is based on a uniform grid. These savings result in substantially lower runtimes.

The coarse approximation of far-away obstacles was motivated by the limited precision of robot-based sensing for far-away objects and by the larger obstacle movements that are possible before the robot comes close to them. If these conditions are met, the generated paths have similar quality as the ones generated using a grid of uniformly high resolution.

Since the runtime of the multiresolutional path planner is very short, it can be used for continuous replanning. This is not wasteful, since only the initial part of the path that is executed immediately after planning is planned in detail.

An example with a two-dimensional configuration space has been presented. The generated trajectories facilitated the fast movement towards the target while at the same time minimizing the chances of collisions.

Furthermore, it has been shown, how to include time into the configuration space. This makes planning with moving obstacles possible. The non-uniform sampling of the time-dimension leads only to a moderate increase in computational costs.

So far, the kinematics of the robot has not been included in the configuration space. Since the running time of the planner is only a few milliseconds long, it would be feasible to increase the dimensionality of the configuration space and still replan at a high rate. One could e.g. explicitly model the orientation or the velocity of the robot. This will be subject to future research.

References

1. R. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. In *Proceedings of the 8th International Conference on Artificial Intelligence (ICAI)*, pages 799–806, 1983.

2. James Bruce and Maria Manuela Veloso. Real-time randomized path planning for robot navigation. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '02)*, October 2002.
3. B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, 1993.
4. P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths in graphs. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100–107, 1968.
5. S. Kambhampati and L.S. Davis. Multiresolution path planning for mobile robots. *IEEE Journal of Robotics and Automation*, RA-2(3):135–145, 1986.
6. L. Kavraki, P. Svestka, J.C. Latombe, and M. Overmars. Probabilistic road maps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.
7. O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotic Research*, 5(1):90–98, 1986.
8. J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.
9. J.-C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *International Journal of Robotics Research*, 18(11):1119–1128, 1999.
10. T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, 1983.
11. N.J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, pages 509–520, Washington, DC, 1969.
12. C. O'Dunlaing and C. K. Yap. A 'retraction' method for planning the motion of a disc. *Journal of Algorithms*, 6:104–111, 1986.