International Conference on Intelligent Robots and Systems in San Diego, USA, 29 October – 2 November 2007

Path Planning and Execution in Fast-Changing Environments with Known and Unknown Obstacles

Thorsten Gecks and Dominik Henrich

Lehrstuhl für Angewandte Informatik III (Robotik und Eingebettete Systeme)
Universität Bayreuth, D-95445 Bayreuth, Germany
E-Mail: {thorsten.gecks, dominik.henrich}@uni-bayreuth.de
http://ai3.inf.uni-bayreuth.de/

Abstract **– We present a path planner capable of efficient and real-time handling of known and unknown obstacles in highly dynamic workspaces. Known obstacles are acquired offline and stored in a world model, unknown obstacles are acquired online by one or multiple sensors. This is a typical situation for many applications. The method presented here exploits this distinction by building a static roadmap based on known obstacle information. This enables efficient path planning and real-time performance using bounded lazy evaluation thus reducing the number of costly collision test. The dynamics of the workspace are addressed by invalidation/revalidation of roadmap edges based on sensoric input. Several revalidation strategies are evaluated. The proposed path planner is probabilistically complete and utilizes global environment information to assure goal arrival, if the goal is reachable. Our approach is realized using standard PC hardware with computational requirements allowing real-time performance. Experimental results show the validity of our approach.**

Index Terms – motion planning, multisensor systems, robots

I. Introduction

To motivate the problems addressed by this path planner concept, we will start by giving a typical real-world example. Based on the example, we will categorize path planning problems into four categories and show how the current study fits into these categories.

One typical task would be the navigation of a car through an urban scenario based on a street map. This map provides enough information to generate an optimal path in time or space. During execution of this path unforeseen hindrances may appear (traffic jams, accidents, etc.) because the environment may change faster than the planning and execution of the desired path can be carried out. Thus, the problem is characterized by two properties: The first property describes whether the complete obstacle information needed for the collision test is available before runtime. The second property relates to the combination of the computational costs in terms of the runtime of all tasks involved in collision-free movements, when performed in a sequential manner. The costs include: the time t_{sens} needed for sensor acquisition and processing, the time t_{plan} needed for planning and the time t_{exec} needed for the execution of the planned path. The sum of these runtimes can be compared to the time period t_{env}, during which the environment can be considered static with respect to the given planning problem. Table I gives an overview of the mentioned categories and sample applications for each field.

TABLE I
CATEGORIZATION OF PATH PLANNING PROBLEMS AND EXAMPLE APPLICATIONS

	Time relationships	
	$t_{sens} + t_{plan} + t_{exec} < t_{env}$	$t_{sens} + t_{plan} + t_{exec} > t_{env}$
O_{known}	NC milling, sweeping	Kinodynamic motion planning
$O_{unknown}$	Path planning with local sensors, medical robotics	Human-robot cooperation, service robotics

The first category of planning problems comprises environments O_{known} that are completely known in advance with no time constraints for planning and execution. Examples of applications include the well-known field of numerical control (NC) tool path generation for milling [4] or sweeping algorithms covering a given static area [16], for example used by lawn mower robots.

The second category encompasses all planning problems where known environmental dynamics are a considerable factor and have to be considered explicitly in the planning process. Kinodynamic motion planning [3] can provide a solution to such problems. Typical applications for this type of motion planner are aerospace tasks, e.g. spacecraft rendezvous.

In the third category, the environment $O_{unknown}$ is unknown and sensor input is necessary to allow for collision tests with obstacles in the environment. Additionally, the time required for planning is not critical, as the environment is static for long periods of time. In medical robotics for example, the information about the operative site is often acquired in a preoperative phase, e.g. with tomography. With this information, complete, optimized planning and execution of the milling of a cochlear implant is possible without further acquisition of environmental information, as described in [15]. Planning with local sensors is similar. The planning only comprises the next immediate action due to the limited range of short-range distance sensors, such as the capacitance sensors used in [8] and [14]. Potential field planning methods, as described for example in [13], are a very common solution for this type of sensor. Typically, they assume the environment to be static for the next incremental planning step.

The fourth category consists of fast-changing environments containing unknown obstacles to be avoided, making continuous sensor updates necessary to ensure safe path planning. The path planning algorithm must adapt to environmental changes and guarantee sufficiently short runtimes to enable a decent sensor update rate. Performing a complete planning and execution is not feasible, because the environment may change before the goal is reached. Examples of this category include human robot cooperation in industrial environments or the navigation of a mobile robot through a building with dynamic obstacles, such as crowds of people. The path planning algorithm described in this paper provides a solution for planning problems of category four. Additionally we require the planner to solve high-dimensional planning problems for robots with many degrees of freedom.

The following Section II describes the state of the art in efficient path planning for dynamic environments, followed by Sections III and IV describing our approach to the path planning problem. Section V gives a short overview of the robotic system realized with this algorithm and presents experimental results demonstrating the real-time capabilities of this approach. Section VI contains concluding remarks about the work done.

II. Related work

In this section, we give an overview of existing approaches to reduce planning costs. The basic assumption is an expensive collision test compared to the other components of path planning (like roadmap construction, etc.).

As a consequence of the global collision detection described above, global path planning is a natural solution exploiting all available information and is complete even in cluttered situations that would lead local path planners into traps (local minima). With regard to the dimensionality of the configuration space, randomized roadmap planners scale with dimension without major drawbacks in computational efficiency. Several optimizations have been proposed in the past that lower the computational costs, specifically by reducing the number of collision tests or the required enviroment detail. In the following, we focus on reducing the number of collision tests to address systems with costly collision tests (i.e., in our system around 1 ms for a single robot position).

A significant reduction in the number of collision tests can be achieved by performing them only when they are absolutely necessary; this is called lazy evaluation. In [2] a roadmap is initially constructed and a path is planned without any collision tests. When a path is found, it is checked for collision in steps from the start and goal node in an alternating fashion. The algorithm stops at the first colliding edge, invalidates it and then starts searching for a new path. However, this algorithm requires a completely known environment. In a fast-changing environment, checking from the goal node towards the start node is inefficient, as the time needed to reach the goal would invalidate the collision test information used for these tests. Another drawback is that the algorithm is not bounded with respect to execution time.

In [9], a similar algorithm is described that determines the shortest path through a given roadmap. Afterwards, this path is tested completely. The collision test information is then used to alter the roadmap at colliding edges and nodes. While it is questionable to test the complete path within a dynamic environment, it also remains unclear why the invalidated edges must be modified, as changes in the environment may revalidate them in the near future. No estimate of the computational cost of a single collision test or the number of collision tests needed for this method is provided.

Another planner using lazy edge evaluation is described in [12]. Based on a static roadmap, the key issue described there is the mapping of real-world changes to invalidated nodes of the roadmap via a table lookup. The table maps occupied voxels to roadmap nodes. These occupied voxels are calculated from a model of the known objects in the workspace and the table is constructed from a given robot model in an offline step. Besides the missing obstacle feedback from sensors, the size ($1.60\times2.44\times1.36$ m^3) and granularity (4 cm resolution) of the rasterized workspace are rather limited due to computational restrictions. For a network of 5000 nodes, the planning time achieved was 74 ms on average on a 2.8 GHz Pentium IV PC. With 10.000 nodes this increased to 385 ms.

In [1], a combination of methods is presented to provide real-time path planning capabilities for a mobile robot in a partially known environment consisting of a landscape with buildings in known positions and unknown, autonomous agents detected by the robot's laser scanner. An initial, static roadmap is built based on the known environment information. The shortest path is generated through this roadmap and updated during execution based on the obstacles encountered on the way. The path is updated with the newly developed graph planning algorithm, Anytime Dynamic A* [11]. In addition to the limited local sensor information, the results achieved on a PC with up-to-date hardware would not allow the algorithm to be applicable to fast-changing environments, as the maximum planning time peaks at 0.2 seconds. Also, the collision test implemented in this algorithm was inexpensive, testing more than 50,000 edges in 0.1 seconds, which is far less than the collision test costs assumed here.

In summary, we can state that lazy evaluation is a key to the solution but the concept itself is too general to provide real-time capabilities. It must be modified to specifically meet these requirements. Furthermore, the distinction into the two types of collision tests (known and unknown obstacles) typical for many applications needs to be addressed and exploited appropriately.

III. Algorithm Description

In this section, we present our path planning algorithm in detail. All functions left unspecified in pseudo code are explained in text. For simplicity, the revalidation of invalidated roadmap edges is postponed to the subsequent section.

The proposed path planning algorithm presented is based on a static randomized roadmap containing a set of N

vertices V representing robot configurations in joint space and a set of edges E representing direct connections of vertices (either linear connections or solved by a local planner). This roadmap is constructed in an offline step using the function *init*().

```
$\textbf{\textit{init}}()$
    $\textbf{for } i=1 \textbf{ to } N$
        $addVertex(V)$
    $\textbf{forall } v \in V$
        $connect(v, k)$
```

The function *addVertex*(V) extends V with a vertex that is assured to be collision-free with regard to the known obstacles. Several methods exist for the generation of vertices [6]: Uniform generation places vertices within the workspace using random or pseudorandom methods, non-uniform generation takes into account the current obstacle situation and, for example, places vertices close to the border between obstacles and free space, which is expected to solve planning problems by increasing vertex density in this problematic area. After the initial number of vertices has been generated, *init*() interconnects these vertices using the function *connect*(v,k):

```
$\textbf{\textit{connect}}(v, k)$
    $S = getKNearestNeighbours(v, k)$
    $\textbf{forall } s \in S$
        $\textbf{if } (\textbf{not } collision(O_{known}, (v, s)))$
            $E = E \cup \{(v, s)\}$
```

The function *connect*(v, k) connects a given vertex v to a set S of k nearest neighbors delivered by *getKNearestNeighbours*(v, k). This ensures that the costs of vertex expansion are limited when searching for the shortest path through the roadmap. For each of the k nearest-neighbor-vertices, the direct connection to the given vertex v is checked for collisions with known obstacles using the function *collision*(O,x), returning True if a collision occurs and False otherwise. O represents obstacle information with which collisions can be detected and x is either an edge or a node. Two types of obstacle information can be distinguished, O_{known} and $O_{unknown}$. The set O_{known} contains obstacles known before runtime of the path planner. In our prototype these are represented by a B-rep of the machinery and layout of the robot workspace and collisions are detected using standard GJK-Algorithm [7]. The set $O_{unknown}$ contains unknown obstacle information acquired by sensors at runtime. Vertices and edges contained in the static roadmap are thus already tested against known obstacles, which reduces collision test costs online.

```
$\textbf{\textit{planPath}}(v_{curr}, v_g)$:
    $connect(v_{curr}, k)$
    $connect(v_g, k)$
    $\textbf{while}(v_{curr} \neq v_g)$
        $P = searchShortestPath(v_{curr}, v_g)$
        $\textbf{if}(P \neq \emptyset)$
            $executePath(P)$
        $\textbf{else if}(\textbf{not } collision(O_{unknown}, v_g))$
            $addVertex(V)$
```

Online path planning is performed by *planPath*(v_{curr},v_g) and *executePath*(P) in an interleaved manner. After a valid path is found, *planPath*(v_{curr},v_g) invokes *executePath*(P), which executes and evaluates the path concurrently.

Initially, *planPath*(v_{curr},v_g) connects the start configuration v_{curr} (the current robot configuration) and the given goal configuration v_g to the static roadmap. In many applications, a small set of start and end points for robot motions occurs repeatedly, thus adding these points to the roadmap is feasible.

The function *searchShortestPath*(v_{curr},v_g) returns a path P computed using a shortest path search in the static roadmap with standard algorithms such as A* or Dynamic Anytime A*[11]. Then, if a path exists, it is executed, otherwise the roadmap needs to be extended with vertices, but only if the goal node is collision-free with respect to unknown obstacles, because adding vertices would not help find a valid path otherwise.

Based on the path found, the system immediately begins driving the robot accordingly and testing collisions with unknown obstacles for the edge currently being followed. For clarity, this look-ahead is simplified to a *collision*($O_{unknown}$,(v_1,v_2)) call in the *executePath*(P). In our prototype system, the look-ahead is realized as follows: The edges of the path found are subdivided into discrete steps. A constant number of steps is checked in advance. The number of steps tested is limited by the runtime available for the collision test.

```
$\textbf{\textit{executePath}}(P)$
    $\textbf{forall } e_i = (v_1, v_2)_i \in P$
        $\textbf{while}(v_{curr} \neq v_2)$
            $\textbf{if } (collision(O_{unknown}, (v_{curr}, v_2)))$
                $setInvalid(e_i)$
                $V = \{v_{curr}\} \cup V$
                $connect(v_{curr}, k)$
                $\textbf{return}$
            $\textbf{else}$
                $v_{curr} = driveRobot(v_{curr}, v_2, \Delta t)$
```

If the collision test indicates a collision-free path, the robot is driven along the path for a certain amount of time. In case the collision test indicates a collision along the current edge, the edge needs to be invalidated. The function *setInvalid*(e) flags the edge e, so that the edge will not be considered in *searchShortestPath*() the next time it is invoked. This way, the dynamics of the workcell are efficiently mapped to the static roadmap, producing a dynamic roadmap. All invalid edges are repeatedly tested for revalidation, as described in the following section. The current vertex v_{curr} is then inserted into the roadmap and connected to its neighbors. This includes the assumption, that dynamic obstacles frequently reappear at the same places. After that, *executePath*(P) returns and a new cycle begins, comprising a planning step on the static roadmap excluding invalidated edges.

IV. Edge Revalidation

In this section, we examine how invalidated roadmap edges could be revalidated efficiently and adapted to the changes in the environmental situation.

Edges invalidated due to unknown obstacles should not remain invalidated forever. In fast-changing environments, this would lead to the invalidation of large parts of the roadmap and thus would produce inefficient paths and unnecessary addition of roadmap vertices. In Table II, a number of possible edge revalidation strategies are compared and explained in the following paragraphs. The table is sorted in ascending order with respect to computational costs.

Table II
Comparison of Edge Revalidation Strategies (ERS)

RS (nr.)	Advantages	Disadvantages
Never (1)	No computational costs	Inefficient roadmaps
When goal is reached (2)	Very low computational costs	Inefficient roadmaps
Timeout (3)	Low computat. costs	Local traps possible
Sensor-indicated (4)	Adaptation to environmental changes	Suboptimal adaption
Sensor-determined (5)	Optimal adaptation to environment changes	High computational costs

The “Never” strategy keeps edges invalidated forever. This is equivalent to deleting edges from the roadmap. Searching for shortest paths would always consider obstacles that, in a fast-changing environment, are typically no longer present. The path planner would thus generate inefficient paths avoiding non-existent obstacles, if it is able to find a path at all. If not, it would have to add edges to the roadmap, which is a rather expensive operation due to the collision tests required. The collision test information with known obstacles present in the invalidated edges is discarded and never used again.

The “When goal is reached” strategy is another simple technique. It revalidates all invalidated edges upon termination of the current planning and execution process, that is, when the goal is reached. Nevertheless, in environments with fast-moving obstacles, this may also lead to long-lasting widespread edge invalidation, inducing the same problems as in the “Never” strategy. In environments with static obstacles, this strategy produces unnecessary collision tests.

A strategy with comparatively low computational costs is the “Timeout” strategy. After invalidation, each edge is assigned a timeout after which it is again revalidated. In environments where a certain minimum obstacle speed v_{min} can be assumed, this is a feasible solution as it revalidates the edges that could be revalidated anyway because the related obstacle volume has moved. Nevertheless, this strategy can lead to cyclical behavior (which was verified in our experiments) in the presence of static obstacles. With a timeout-based strategy, edges are revalidated although they are still invalid due to the static obstacles. If *searchShortestPath*(.) then finds a path through these edges, the robot will return to a position it has been before and start invalidating edges again, resulting in cyclical behavior. Increasing the timeout may be a solution but this biases the strategy to the aforementioned strategies. Selecting an optimal timeout for any kind of environment is not possible.

The “Sensor-determined” strategy relates to an optimal revalidation of edges, where mapping from the obstacle volume to roadmap edges or vertices exists, as proposed in [12]. Invalidated edges could then be revalidated immediately after the associated obstacle volume becomes free space. However, as already mentioned in Section II, this would make large amounts of memory necessary for a reasonable robot workspace.

We propose a “Sensor-indicated” strategy based on an abstract sensor model. The sensor can distinguish three basic types of changes in obstacle volumes: increase, decrease and no change in volumes. It is unimportant where these changes occur, as the sensor only indicates one of the three types. Edges are then revalidated as soon as a decrease in volumes is detected. A certain fraction of the invalidated edges is revalidated; thereby several strategies are available: random selection, first-in-first-out, etc.. Although this is suboptimal compared to the “Sensor-determined” strategy, the computational costs are reasonable and for practical purposes, this strategy yields good results. Especially the “static-obstacle-problem” described above is addressed and cyclical behavior is suppressed. On the other hand, continuously moving obstacles such as humans induce a concurrent increase and decrease of obstacle volumes, allowing for short-term revalidation of edges. This in effect leads to a low number of invalidated edges in the roadmap, improving robot mobility in fast-changing environments.

In summary, the “Sensor-indicated” strategy provides a solution for static and dynamic obstacle situations at affordable costs. Our experimental results support this argument (Section V). Mixed scenarios are typical for industrial environments, where for example a maintenance worker carries tools or toolboxes he places at certain points for extended periods. A revalidation strategy should address both the static and dynamic scenarios simultaneously and efficiently.

V. Experiments

The following provides a short overview of the prototype system with which the algorithm was tested. The system deals with human-robot coexistence or cooperation.

A. *Environment*

The collision test is subdivided into two parts: collisions with known obstacles and collisions with unknown obstacles. The first type of collision is detected by a world model describing the various parts of the industrial process, e.g. based on a CAD model of the workspace. The second type is detected using sensory feedback from the environment.

To do so, we developed a camera sensor network observing a common space comprising the complete robot workspace (Fig. 1). Based on difference classification algorithms, these cameras deliver the current dynamic obstacles in the workspace marked as *foreground* pixels in each camera. All other pixels are marked as *background*.

By the means of a robot model, the system can avoid collisions by projecting future robot configurations into the calibrated camera images. Each of these robot images is then checked for proximity to the given foreground pixel set in each respective camera.

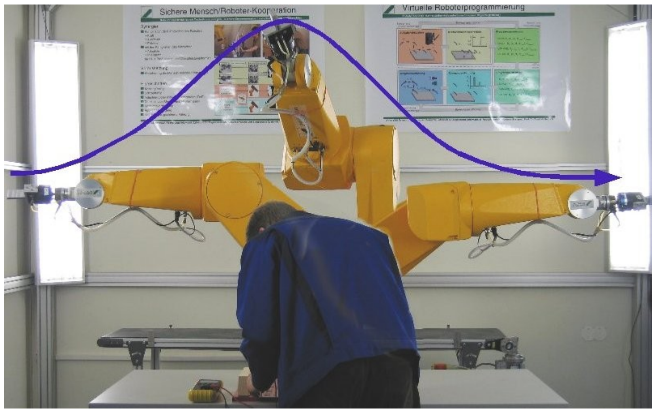
Figure 1: View of the workspace with illumination and cameras attached to an aluminium cage. The collision-avoiding robot path is indicated by the blue arrow. The undisturbed motion would be the direct connection between the left and right endpoints.

The distances from all cameras are fused (with respect to object occlusions) to form an overall collision statement as described in [5] and [10]. This image-based collision test provides collision information in 3D and can be calculated online in real-time. A collision test for a single robot configuration in an eight-camera network takes one millisecond, including sensor fusion. While this is quite fast, the number of collision tests required for a linear path in the configuration space can be quite large, as this path has to be interpolated by a number of configurations. The number of collision tests is therefore limited by the imminent real-time demands. With the distance d_{obst} determined as the current distance to the closest obstacle and given a certain robot speed $v_{robot} > 0$ and a maximally achieved obstacle speed v_{obst}, the time-to-collision t_{coll} is given by

$$t_{coll} \geq \frac{d_{obst}}{v_{robot} + v_{obstacle}} \qquad (1)$$

This is equivalent to the time t_{env}, during which the environment can be assumed to be static, as mentioned in Section I. For our human-robot coexistence system, the time t_{env} is fixed to 100ms and the robot speed v_{robot} is controlled with respect to the current obstacle distance (equation 1 solved for v_{robot}). Below a threshold minimum distance to obstacles that would result in a negative v_{robot}, the robot is halted and thus the planner must not be activated. Edge revalidation is realized using the "sensor-indicated" strategy based on foreground pixel information (Section IV).

B. *Experimental Setup*

In the experiments we concentrated on dynamic aspects of the enviroment, as static path planning benchmarks such as SIMPLE, TRAP, DETOUR, etc. have been investigated largely for PRM-planners in the past. Thus we concentrate on dynamic aspects and choose typical real-world scenarios for simulation.

The following experiments consist of a simulated obstacle scenario to assure reproducibility and fairness of comparative measurements and to test the validity of the speedup assumption in the static obstacle case. The collision test was performed in 3D on the simulated objects. The experiment ran on a standard PC workstation (AMD X2-3800+, 2GB RAM).

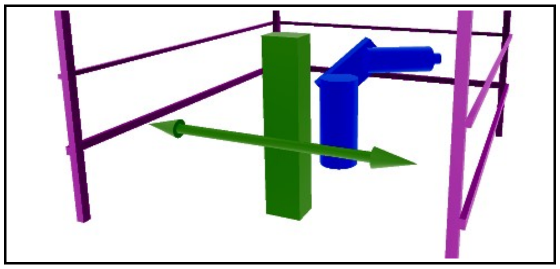
Figure 2: Simulated "Moving obstacle" scenario. The green cuboid obstacle in the front moves cyclically from left to right with increasing speed. Without interruption the robot would take a path that intersects the path of the obstacle.

Real world experiments were also conducted for evaluation. They included a robot controller (Stäubli CS7-B) and an ethernet network providing the communication required to control the robot. In the real world example with sensory feedback, we achieved an update rate of 10-15 Hz (Fig. 1) at a robot speed of 0-80 cm per second and covered a workspace of approximately 4x4x3 m with a resolution less than 6 cm.

We simulated several scenarios and evaluated each with the different revalidation strategies mentioned in Section IV. The static roadmap initially contained 4192 vertices with each vertex connected to 26 neighboring vertices. Table III lists the runtimes of the two most calculation-intensive algorithm sections common to all scenarios. The runtime for the collision test mentioned in Table III constitutes the sensor-based collision test. Additionally the image processing speed is given to explain the update rate achieved.

Table III
Average Runtime of Central Algorithm Parts (with Standard Deviation in Parantheses)

Edge collision test with unknown obstacles [*collision*($O_{unknown}$, *e*)], 3289 samples	16.6 (2.4) ms
searchShortestPath(v_{curr},v_g), 50 samples	0.85 (0.21) ms
Sensor data acquisition (4 camera-system)	70 ms

The robot movement simulated in all scenarios consisted of repeated movement back and forth between two targets combined with path planning to avoid any obstacles. This cycle was repeated several times to duplicate the cyclical behavior of robots in industrial environments.

The first scenario is given for reference, consisting of the robot moving through the empty space with no obstacles. The variable e_{avg} describes the average number of invalidated edges when *searchShortestPath*(.) is called. This is an indicator for the mobility of the robot. The total number of edges ever invalidated E_{inv} indicates how often the path planner tried a path obstructed by an obstacle. The path length for each path from one target to the other in configuration space l_{avg} is calculated as the sum of all joint angle differences arising along the path and is averaged over all cycles.

Table IV
Simulated Results. The first column lists scenario (S), the first row lists the revalidation algorithm (RA) in columns 3-6

S	Measurements	RA 1	RA 2	RA 3	RA 4
1	e_{avg}	0	0	0	0
	E_{inv}	0	0	0	0
	l_{avg}	123.5°	122.6°	122.6°	122.6°
2	e_{avg}	770.8	468.3	361.4	770.9
	E_{inv}	838	9988	9916	1006
	l_{avg}	209.5°	215.3°	214.0°	209.3°
3	e_{avg}	794.48	107.3	87.6	131.2
	E_{inv}	838	2480	2510	1007
	l_{avg}	210.5°	147.2°	147.0°	151.4°
4	e_{avg}	1271.8	446.9	410.3	15.3395
	E_{inv}	1540	13038	26317	27309
	l_{avg}	261.9°	279.5°	286.0°	202.02

The second scenario comprises a single static object within the robot's path. Interesting are the large values of E_{inv} for revalidation Strategies 2 and 3. These extreme values were caused by the frequent revalidation of edges, which is not necessary in this scenario.

Scenario 3 consists of the same static object found in Scenario 2, but in this case the object vanishes after a short time period. Here we can see the benefit of Strategy 4, in that it provides a low l_{avg}, while maintaining a low E_{inv}. The e_{avg} of Strategies 2 and 3 is better, but at the cost of unnecessary testing of edges that are obstructed.

Scenario 4 is depicted in Fig. 2. The robot moves a longer distance, so that l_{avg} is not directly comparable to the other scenarios. Here we can see that Strategy 4 provides short robot paths resulting in rather high E_{inv}.

A combination of scenarios 3 and 4 is very common in the real world, indicating that strategy 4 is the best solution, as it yields good or the best performance in both scenarios.

VI. Conclusion

We presented a new path planning approach for a category of path planning problems with real-time demands and an expensive collision test. The collision test is separated into tests with known and unknown obstacles, which is feasible for many real-world applications. The algorithm exploits this separation by using a static roadmap and addresses the real-time demand with bounded lazy evaluation. Planned edges of the roadmap are tested and invalidated if necessary, thus dynamically adapting the roadmap to the current environmental situation. Additionally, we presented a framework for efficient adaptation of the static roadmap to quickly changing environments. Therefore, we investigated and tested several edge revalidation strategies. Our experimental results show that real-time performance is possible together with responsive adaptation to workcell dynamics using appropriate revalidation strategies.

In the future, point placement strategies as described in [6] must be investigated and the comparison of shortest path algorithms with regard to their effects on computational costs should be evaluated.

References

[1] J. van den Berg, D. Ferguson, J. Kuffner: “Anytime Path Planning and Replanning in Dynamic Environments”, Proceedings of the 2006 IEEE International Conference Robotics and Automation, May 15-19, 2006, pp. 2366- 2371, Orlando, USA, 2006

[2] R. Bohlin, L.E.Kavraki: “Path Planning Using Lazy PRM”, Proceedings of the 2004 IEEE International Conference on Robotics and Automation, 521-528 vol.1, April 24-28, 2000, San Francisco, USA

[3] B. Donald, P. Xavier, J. Canny, J. Reif: “Kinodynamic motion planning”, Journal of the ACM, vol. 40 no. 5, pp. 1048-1066, 1993

[4] D. Dragomatz, S. Mann: “A classified bibliography of literature on NC milling path generation”, Computer-Aided Design, Vol. 29 No. 3, pp.239-247, 1997 Elsevier Science Ltd

[5] T. Gecks, D. Henrich: “Multi-Camera Collision Detection allowing for Object Occlusions”, In: 37th International Symposium on Robotics (ISR 2006) / 4th German Conference on Robotics (Robotik 2006), München, Germany, May 15-17, 2006.

[6] R.Geraerts, M. Overmars: “Sampling Techniques for Probabilistic Roadmap Planners”, Technical report UU-CS-2003-041, Institute of information and computing sciences, Utrecht University

[7] E. G. Gilbert, D. W. Johnson and S. S. Keerthi: “A Fast Procedure for Computing the Distance between Complex Objects in Three-Dimensional Space”, IEEE Transactions on Robotics and Automation 4(2): pp. 193-203, April 1988

[8] J.T. Feddema, J.L. Novak: “Whole Arm Obstacle Avoidance for Teleoperated Robots”, Proceedings of the 1994 IEEE International Conference on Robotics and Automation, pp. 3303-3309 vol.4, San Diego, CA, USA, 8-13 May 1994

[9] L. Jaillet, T. Simeon, “A PRM-based Motion Planner for Dynamically Chaning Environments”, Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai International Center, Sendai, Japan September 28 - October 2, 2004

[10] S. Kuhn, T. Gecks, D. Henrich: “Velocity control for safe robot guidance based on fused vision and force/torque data". In: IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, pp. 485-492, Heidelberg, Germany, Sep 3-6, 2006.

[11] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun: “Anytime Dynamic A*: An Anytime, Replanning Algorithm”, Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), Monterey, USA, June 2005

[12] H. Liu, X. Deng, H. Zha, D. Ding, “A Path Planner in Changing Environments by Using W-C Nodes Mapping Couples with Lazy Edges Evaluation”, Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 9-15, Beijing, China

[13] V. Lumelsky, E. Cheung: “Real-Time Collision Avoidance in Teleoperated Whole-Sensitive Robot Arm Manipulators". In: IEEE Transactions on Systems, Man and Cybernetics, Vol.23 No.1, pp.194-203,1993.

[14] J.L. Novak, J.T. Feddema: “A Capacitance-Based Proximity Sensor for Whole Arm Obstacle Avoidance". In: IEEE Proceedings of the International Conference on Robotics and Automation, pp. 1307-1314, 1992, Nice, France

[15] M. Waringo, D. Henrich: “3-dimensionale schichtweise Bahnplanung für Any-Time-Fräsanwendungen”, Robotik 2004, 17.-18. June 2004, VDI-Berichte 1841, pp. 781-788, Munich, Germany

[16] A. Zelinsky, R.A. Jarvis, J.C. Byrne, and S. Yuta: “Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot. In Proceedings of the International Conference on Advanced Robotics, pages pp533--pp538, Tokyo, Japan, November 1993