In GWR09 German Workshop on Robotics, Braunschweig, Germany, June 9-10, 2009

Sensor-based Online Planning of Time-optimized Paths in Dynamic Environments

Thorsten Gecks and Dominik Henrich

Lehrstuhl für Angewandte Informatik III, Universität Bayreuth, Germany
{thorsten.gecks|dominik.henrich}@uni-bayreuth.de, http://www.ai3.uni-bayreuth.de

Abstract Dynamic environments, in which robots and for example humans share a common workspace impose a variety of requirements on path planning algorithms, including real-time capabilities and collision tests based on sensor input. We propose a randomized-roadmap-based path planning algorithm that limits the number of collision tests and distance calculations to a volume achievable in real-time, while still being able to achieve high path clearance and statistical completeness given an unlimited number of planning cycles. It does so by exploiting the knowledge about static portions of the environment using a static, collision-checked roadmap and by interleaving planning and execution. Image-based distance measurements are induced by the graph search algorithm and interpolated to allow planning of time-optimized paths on the roadmap with a reduced number of distance measurements.

1. Introduction

Robots are emerging from their stereotyped role as dumb, mechanical tools in separated environments and becoming user-friendly, cooperative workmates. Thus, sensible reactions to dynamic environments are becoming a predominant issue. Safety and intuitive usability during user interaction are important aspects, as well as efficiency and availability on the process side. A building block for these kinds of robots is the ability to detect and cope with objects in the robot's workspace in a reactive fashion.

Planning in the presence of dynamic obstacles is a challenging problem, especially if these obstacles are unknown in advance, meaning that the information about their location and geometry has to be acquired by sensors. For robots with a high degree of freedom, it is hard to explicitly translate workspace obstacles into configuration space in order to reconstruct the free/occupied configuration space as a whole. In consequence, collision tests or distance measurements to obstacles are time-consuming for each robot configuration.

The demand for a high-clearance planning method is generated from two sides: a psychological component and a safety component. Humans do not feel comfortable when faced with a speedy, heavy-duty robot arm in close proximity. This becomes even more relevant, when the worker is performing a different task requiring his full attention and, due to distraction, no immediate reaction to erroneous robot movements is possible. The safety component requires the robot to reduce its speed in the vicinity of dynamic obstacles so it could react safely to unforeseen obstacle movements within the remaining time to collision. This distance-based robot speed regulation generally favors high-clearance paths for time-optimized path planning.

2. State of the art

Distance-optimized or maximum-clearance path planning has long been a field of research that has produced a variety of methods. Principally, these methods explore a local portion or the global, complete workspace or configuration space of the robot, computing a path from the start to the goal, while maintaining a great distance to all obstacles all along its way.

The most reactive and real-time capable algorithms are the family of potential field methods [Kathib86]. Simulated potential fields allow the calculation of repulsive forces that can be translated into robot movements. While several extensions exist, this method is generally susceptible to local minima that prevent the robot from reaching the goal. Those extensions commonly rely on a specific modelling and/or a complete knowledge of obstacles in the robot configuration space [Barraquand91].

Another family of methods relies on geometric information about obstacles (either in workspace or configuration space), that is easy to decompose in that it relies on simple geometric primitives (i.e. B-Reps with planar surfaces). Based on a decomposition of the free space, it is possible by searching the connectivity graph to find a path from the start to the goal with maximized clearance [Schwartz83]. This technique can be combined with the previously mentioned potential fields if each decomposed cell contains a local potential field that repels the robot away from the cell boundaries, while still allowing it to react to dynamic obstacles [Lindemann05].

A combination of artificial forces and roadmap-based planning was presented in [Brock99, Quinlan93]. The elastic strip method initially uses an arbitrary planner to produce an optimal path from the start to the goal. Upon execution, the path is deformed by any moving dynamic obstacles due to artificial repulsive forces associated with those obstacles. While this approach is real-time capable and complete, the path can become highly inefficient if obstacles deform it from the initial state to such an extent that re-planning becomes necessary. Also, paths that were

originally obstructed can become viable if obstacles move, but are not considered after the initial path planning.

For the class of probabilistic roadmap planners, several algorithms have been proposed to either sample configurations on the medial axis of the free space [Holleman00] or to retract the path to the medial axis in a post-processing step after planning through a standard probabilistic roadmap [Geraerts04]. Both methods rely on detailed knowledge of the obstacle surfaces in the planning space.

As we have seen from this section, the approaches presented so far either provide real time reaction to dynamic obstacles but can become caught in local minima (and thus are incomplete) or rely on the exact knowledge of the configuration space or the execution of a high number of collision tests, which is impractical for robots with a high degree of freedom. In the following section we will outline a path planning algorithm that is able to provide real time path planning while generating high-clearance paths in dynamic environments.

3. Real-time shortest-path planning algorithm

In this section, we will give a short overview of the basic techniques that we use as a basis for path planning in dynamic environments. Section 4 extends these techniques towards time-optimized path planning. For further details on the following, see [Gecks07].

The basic algorithm described below is real-time-capable under the assumption, that the most computationally intensive part of planning is the collision tests (or distance calculation) and not the graph search. In our case, *each* collision test takes 1…5 ms to compute, while the graph search without collision tests on a reasonably-sized roadmap has an average runtime of well below 10 ms. To separate graph searching and collision tests, the graph search algorithm rests upon an abstract layer, which provides the edge costs determined by the collision test. The number of collision tests/distance calculations used is then adaptable and transparent to the graph search algorithm. Thus, the overall runtime can be tailored to the real-time demands imposed of the application.

The proposed path planning algorithm is based on a static randomized roadmap $G(V, E)$, consisting of a set of vertices V and a set of edges E. Edges and vertices are tested in an offline setup step for collision with a known static environment. Online tests refer to the unknown, dynamic objects in the environment, which are detected by sensors, such as multiple surveillance cameras. As these online tests are relatively expensive, their number is reduced by several techniques such as lazy collision checking and interleaving of planning and execution. The following pseudo code outlines the major components.

In the function *planPath* the start (current) v_{curr} and the goal robot configuration v_g are connected to the roadmap G via *connect*(v), which constructs a new vertex v, if it does not already exist. It then adds collision-tested connections from the

new vertex to its k neighbors. As long as the target is not reached, planning *searchShortestPath*(v_g, v_{curr}) and execution phases *executePath*(P) are interleaved. The function *searchShortestPath*(…) finds the shortest path from v_{curr} to v_g via the roadmap G using the graph searching algorithm A*, if it exists. If no path is returned by *searchShortestPath*(…) and the goal does not collide with unknown obstacles (detected by *collision*($O_{unknown}$, v_{curr})), the function *addVertex*(V) samples a new node (uniform or non-uniform, see [Geraerts04]) and connects it to the roadmap.

```
planPath(v_curr, v_g)
    connect(v_curr)
    connect(v_g)
    while(v_curr ≠ v_g)
        P = searchShortestPath(v_curr, v_g)
        if (P ≠ {})
            executePath(P)
        else if (not collision(O_unknown, v_g))
            addVertex()
```

```
executePath(P)
    while(v_curr ≠ v_g)
        if (testPath(O_unknown, v_curr, P)
            setInvalid(e_curr)
            connect(G, v_curr)
            return
        else
            v_curr = driveRobot(P, Δt)
```

A valid path is executed in the function *executePath*(P). The function *testPath* ($O_{unknown}$, v_{curr}, P) tests the given path P from the current robot position v_{curr} towards the goal. The depth of the test only depends on the available computing power and the given collision test costs. If a collision occurs, the current edge e_{curr} is invalidated and omitted in the next path planning phase. Non-invalidated edges thus form a dynamic subgraph G_{valid}, which incrementally adapts to the current obstacle situation as path execution proceeds. Invalidated edges are revalidated using various methods described and evaluated in [Gecks07]. As a consequence of collision checking before execution, the algorithm is guaranteed to find a collision-free path to the goal and because of the randomized roadmap approach it is statistically complete.

4. Time-optimized path planning

In the context of human-robot cooperation and in general when dynamic obstacles are involved, robot speed must be adapted to the current obstacle distance, thus being able to react to movements of the unknown objects before a collision becomes unavoidable. The planning method described beforehand typically follows the configuration space surfaces of the workspace obstacles, as it searches for the shortest, collision-free path to the goal. In combination with speed regulation this leads to rather long path execution times. Optimizing solely for maximum clear-

ance on the other hand could lead to highly inefficient path planning, producing detours and also resulting in high path execution times, too. Thus, clearance competes with path length. In the following sections we will outline how to solve this conflict and how to meet real-time demands at the same time.

4.1 Basic planning algorithm

As a basic requisite for using the A* graph search, it is necessary to define edge costs. Edge costs define vertex costs $f(v)$ via the sum $g(v)$ of edge costs from the start vertex to the current vertex. Together with an estimation of costs to the target $h(v)$, we get: $f(v) = w \cdot g(v) + (1 - w) \cdot h(v)$, the well known cost function of the A* (here we choose $w = 0.5$). For time-optimized planning, edge costs are defined by the time it takes to travel them (Eq. 1).

$$t(v_a, v_b) = \sum_{i=0}^{L-1} \left(\frac{1}{s(d_\xi(v(u_i), v(u_{i+1})))} \left\| v(u_{i+1}) - v(u_i) \right\| \right) \tag{1}$$

In this general (approximating) term, the edge from v_a to v_b is subdivided into L segments $(v(u_i), v(u_{i+1}))$, and a travel time is calculated for each one by dividing the segment length by the robot speed $s(d_\xi)$, which is derived from the average distance d_ξ to the workspace obstacles along the segment $(v(u_i), v(u_{i+1}))$. The distance-regulated speed may be a linear function (for some $k > 0$) of the form:

$$s(d) = \begin{cases} 0 & \text{if } d < d_0 \\ (d - d_0)k & \text{if } d \in [d_0, d_{\max}] \\ s_{\max} = (d_{\max} - d_0)k & \text{if } d > d_{\max} \end{cases} \tag{2}$$

To address the real-time demands, the presented algorithm needs to reduce costly distance calculations. Thus, the overall number of distance calculations is limited per invocation of the path planning algorithm and unknown distances have to be estimated at the roadmap points and along edges of the roadmap, as the geometry of the configuration space obstacles is only implicitly known through the collision test.

Edges are not subdivided to calculate distances along the edge, as the distances between the subdivision points would have to be estimated again in an infinite recursion, so that there is no obvious reason to do so. The distance function along the edge is thus estimated from the distances determined at its vertices. The following two sections describe the edge distance estimation techniques and the vertex distance estimation from calculated vertices, which is also necessary due to the limitation of distance calculations.

4.2 Distance estimation for roadmap edges

In this section we will simplify all possible distance curves along a roadmap edge to a discrete model, deriving several useful conclusions. The distances in the vertices of the edge from v_a to v_b are given by $d_a = d(v_a)$ and $d_b = d(v_b)$. Given a linear connection in configuration space, the edge can be represented by $v(\lambda) = (1 - \lambda)\ v_a + \lambda\ v_b$. We want to derive the distance curve $d(\lambda)$. The environment is considered to be static, that is, no model of obstacle movements exists, as no model of the obstacles in configuration space exists.

In the proposed model, we subdivide λ into L discrete steps $\Delta\lambda = \lambda / L$. For each step, any point of the robot can move by a certain maximum distance, conservatively approximated by the MAXMOVE method [Henrich98]. Thus, the distance to obstacles can increase or decrease maximally by $\Delta d = \text{MAXMOVE}^{-1}(\Delta\lambda(v_b - v_a))$. The distance distribution starts at $(0, d_a)$ and ends at $(1, d_b)$. The resolution of the grid can be adapted so that $(1, d_b)$ can be represented by a discrete grid point (if $d_b \in \mathbf{Q}$).

Within this model, the possible spectrum of distance curves can be represented by sequences of L additions of the vector $(1, \pm 1)$. To simplify the syntax, we will reduce this to a sequence of L ± 1-steps. Two invariants hold true for all possible sequences:

- The number of steps is fixed and given by the resolution of the grid. Specifically, this means that the sum of the occurances of +1 and −1 in the sequence is constant: $\#(+1) + \#(-1) = L$
- As the endpoint $(1, d_b)$ must be reached, the following equation needs to be satisfied: $\#(+1) - \#(-1) = (d_b - d_a) / \Delta d$

Based on these two invariants, it is clear that the number of +1-steps and −1-steps is constant and each sequence is simply a permutation of any other valid sequence. Following this result, there are two limiting distance curves of maximum distance and minimum distances, as seen in Figure 1.

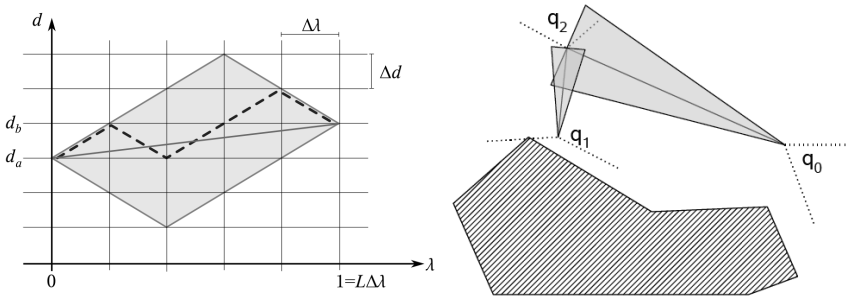

Fig. 1: Left: Model of distance curves. The gray area defines the range of all possible distance curves, the dashed curve represents a discrete example curve. Right: Point distance estimation with estimated distance ranges expanding from calculated distances at point q_0 and q_1 to estimated point q_2.

Thus, at least two distance curve estimations can be derived directly from this model, a *pessimistic edge distance estimation* (the minimum distances curve) and an *optimistic edge distance estimation* (the maximum distances curve). A third estimate that can be derived naturally from this model is the *average edge distance estimation*. In the following, we will derive the equation of the latter estimation.

In the following, n is the number of +1-steps in the sequence and m the number of −1-steps, such that $L = n + m$. The absolute number of all possible curves C_{all} is then given by (selection without repetition, order does not matter):

$$C_{all} = \binom{L}{n} = \frac{L!}{n!(L-n)!} = \frac{L!}{n!m!} \tag{3}$$

For the calculation of the average edge distance estimation, we take a look at the expectation $E_{dist}(i)$ of the distance difference from step i to step i+1:

$$E_{dist}(i) = p_i^+(+\Delta d) + p_i^-(-\Delta d) \tag{4}$$

Where p_i^+ is the probability that a +1-step is at the i-th position and p_i^- the probability for a −1-step. The probability p_i^+ is the number of all curves $C_1(i)$ with a +1 at step i divided by the number of all possible curves C_{all}:

$$p_i^+ = \frac{C_1(i)}{C_{all}} = \frac{\binom{L-1}{n-1}}{\binom{L}{n}} = \frac{\frac{(L-1)!}{(n-1)!m!}}{\frac{L!}{n!m!}} = \frac{(L-1)!n!}{L!(n-1)!} = \frac{n}{L} \tag{5}$$

p_i^+ is thus independent of i. With $p_i^- = m/L$ the resulting E_{dist} is independent of i:

$$E_{dist} = \left(\frac{n}{L}\right)(+\Delta d) + \left(\frac{m}{L}\right)(-\Delta d) = \left(\frac{n-m}{L}\right)\Delta d \tag{5}$$

Thus, the average difference vector in each step is $(\Delta\lambda, E_{dist})$. If we sum up all difference vectors from $(0, d_a)$, we get:

$$d(1) = (0, d_a) + (\Delta\lambda, E_{dist})L = (1, (n-m)\Delta d) = (1, d_a + (d_b - d_a)), \tag{6}$$

which is the endpoint of all curves. So in effect, the average edge distance estimation is a linear function connecting $(0, d_a)$ and $(1, d_b)$.

4.3 Distance estimation in roadmap vertices

The distance $d(v)$ to unknown obstacles in vertex v is estimated based on the distance of the neighbour vertices v_{pred}. Generally, the distance $d(v)$ may vary in the range of $d(v_{pred}) \pm \text{MAXMOVE}^{-1}(v - v_{pred})$. An estimated distance in a roadmap vertex is thus represented as interval $[l, h]$ describing the range within which the true obstacle distance falls.

If a new distance is measured in a vertex, the intervals of estimated neighbour vertices and their successors in the roadmap need to be adapted. As it would be very costly to update all dependent vertices of a roadmap upon distance calculation of a single vertex, the distance estimation of a vertex is updated in the expansion steps of the graph searching algorithm. This may change edge costs, thus the graph search algorithm D* [Likhachev05] is used in a slightly modified form, to update all dependent vertices in the search tree efficiently. Intervals are updated by using the distance information (either calculated or an estimation interval) of connected neighbour vertices, that are "more informed" (see also Figure 1, right). First, the distance information of the neighbouring vertex is propagated along the connecting edge using:

$$[l_{new}, h_{new}] = [l_{pred} - d_{MM}, h_{pred} + d_{MM}] \tag{7}$$

With $d_{MM} = \text{MAXMOVE}^{-1}(v - v_{pred})$. For vertices with a calculated distance d_{calc}, the interval reduces to a point $l = h = d_{calc}$.

Second, the interval of the current vertex v_{cur} (which is $[-\infty, +\infty]$ initially) is updated to the smallest common interval:

$$[l_{cur}, h_{cur}] = [\max(l_{cur}, l_{new}), \min(h_{cur}, h_{new})] \tag{8}$$

For edge cost calculation a single distance needs to be selected from the interval. Three different selection functions can be specified similar to the edge distance estimation: an *optimistic vertex estimation* $f_{dist}(v) = h$, a *pessimistic vertex estimation* $f_{dist}(v) = l$ and an *average vertex estimation* $f_{dist}(v) = (l+h)/2$.

5. Discussion

Translated into a speed, the pessimistic edge or vertex estimations may cause the goal to be unreachable, because the distances are estimated below the minimum speed distance before the goal is reached on any possible path through the roadmap. To prevent this, a speed slightly above zero is chosen as minimum speed in the speed function $s(d)$ for path planning. This is safe, as the path is tested to be collision free before execution and invalidated edges are just skipped by the plan-

ner. If a calculated vertex distance indicates a collision, all edges connected to that vertex are invalidated.

Besides these basic considerations, the chosen estimation technique influences the generated course of the path. The pessimistic vertex estimation biases the planner to prefer shorter paths in estimated regions of the roadmap because of high costs, while the optimistic estimation reduces path costs in the calculated regions of the roadmap, because of low costs in the estimated regions. This leads to obstacle evasion behaviour in the optimistic case and a goal-oriented behaviour in the pessimistic case. This is confirmed by experimental results (Figure 2e, f). For the edge estimations, the pessimistic edge estimator biases the path towards an extended clearance in comparison to the optimistic edge estimator.

In a dynamic environment, the distances calculated become invalid quickly and thus need to be reset before execution of the planning algorithm each system cycle. With only this limited environment information, the path planner can get trapped in local minima or cycle in endless loops. Parameters influencing this behaviour are the number of vertices and edges and the number of available distance tests per execution of the planner. In an environment with more or less static obstacles, this behaviour is undesirable. If the planner can detect whether obstacles move, it can keep distance calculations already done in previous planning cycles. As safety is assured by the underlying collision testing before path execution, this detection does not need to be exact. The planner may only degrade in performance with respect to time-optimized paths when planning on distances that have changed slightly due to small obstacle movements. The sensor system can thus deliver an underestimation of the workspace dynamics, indicating a static environment upon only small obstacle movements.

6. Experimental results

In the following sections we present experimental results for a 2D simulated configuration space and in a real world application with a six-degrees-of-freedom industrial robot. The 2D simulation environment operates the robot using the same parameters (including the distance-speed-function $s(d)$, see Eq. 2) and configuration space size used in the real world application. The collision test and distance calculation are slowed down artificially to resemble the behaviour of the real-world sensor-based counterparts. The simulation environment ensures repeatability of the experiments and is well-suited for visualization of the results. The real-world application shows the feasibility of the approach.

6.1 Simulation of a 2-DOF robot

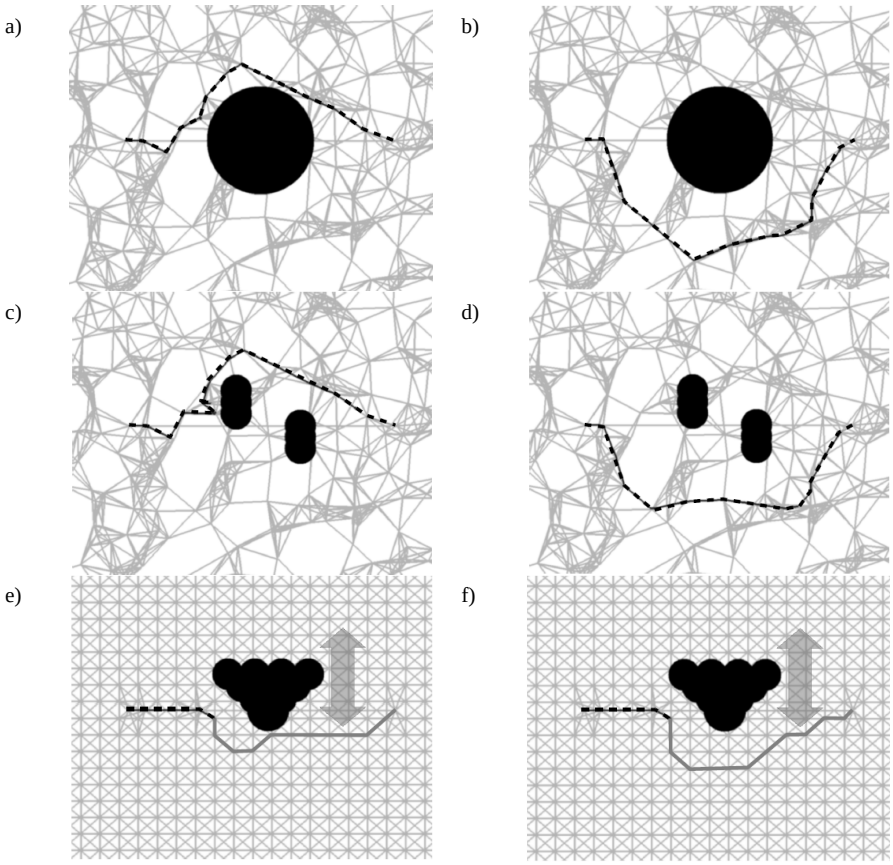

Fig. 2: Robot paths in simulated 2D environments. Executed paths are indicated by dashed lines. For the benchmarks SIMPLE (a,b) and DETOUR (c,d), the shortest path (left) and the time-optimized path (right) are planned in a roadmap with random sampling. The start of the planning task is at the left end of the path, the target at the right. Switchbacks that may be present originate from insufficient planner information about the environment. Images e), f) show a planning step (the robot's progress is denoted by the dashed line and the planned path is indicated by the solid dark gray line) with a moving object (movement indicated by arrow). Both images use a time-optimized planner on a regular grid. e) Uses a pessimistic estimator for roadmap vertices and f) an optimistic estimator. The optimistic estimator biases the graph searching D* to evade the obstacle, while the pessimistic biases it to reduce path lengths to the target.

The simulation experiments were carried out with the shortest-path algorithm outlined in Section 3 and the distance-enhanced algorithm from Section 4 to allow for speedup comparisons. The experiments comprised several standard benchmarks. Some of the setups and results are depicted in Figure 2. All static roadmaps are initialized with either random or regular vertex sampling with two different sizes

(500 and 4000 vertices). Table 1 shows true complete path execution times sorted by benchmark. The planning cycle time of the system is < 100 ms. The time-optimizing planner speeds up the execution time from the start to the goal position by up to 63 percent in the DETOUR benchmark.

Table 1: Benchmark results for shortest-path and time-optimizing planning algorithms, Benchmark SIMPLE (1), SIMPLE with a moving object (2), DETOUR (3). Listed are average *path execution* times in seconds. The planning time in each system cycle is less than 100 ms, thus achieving an update rate of >10 Hz on a 2.2 GHz standard workstation.

		Optimistic Edge Distance Estimation		Pessimistic Edge Distance Estimation		Average Edge Distance Estimation	
Bench-mark	Short-est-Path	Optim. Vertex	Pessim. Vertex	Optim. Vertex	Pessim. Vertex	Optim. Vertex	Pessim. Vertex
1	29.7	13.9	15.7	13.3	15.9	13.9	15.7
2	16.3	12.4	12.7	12.3	11.6	12.0	12.2
3	31.4	13.3	12.9	11.4	12.1	12.8	12.8

6.2 Real-world application with a 6-DOF industrial robot

In this application (Figure 3), the robot speed is scaled up to a maximum of ca. 500mm/s based on its clearance. The system operates at 10 Hz frame rate. The roadmap contains 5000 points, each one connected to ca. 15 neighbouring vertices.

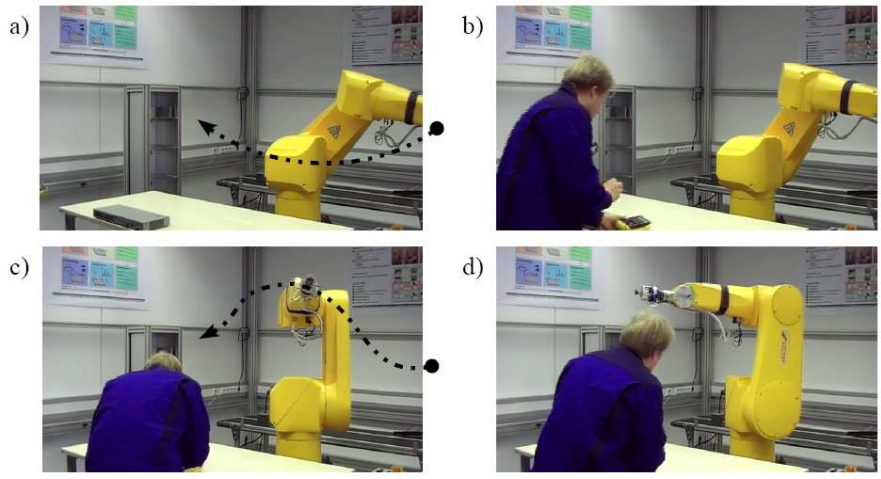

Fig. 3: Robot path in a real-world application. The original robot path is indicated by the dashed line in a). The robot evades a worker entering to perform a maintenance task. The robot executes the path indicated by the dashed line in c).

7. Conclusion

We have presented a time-optimizing path planner for robots with high degree-of-freedom. The planner offers real-time performance by limiting the number of distance calculations and collision tests, while being statistically complete and thus being able to reach the goal even in cluttered environments given an unlimited number of planning cycles. The costs of the collision test are further reduced by offline testing the static environment. The distances are estimated and propagated through the roadmap to benefit from a few calculated distances as much as possible. In the presence of distance-based speed regulation in human-robot coexistence, massive speedups of up to 63 percent compared to the shortest-path solution can be realized.

8. References

[Barraquand91] J. Barraquand and J.-C. Latombe: "Robot motion planning: A distributed representation approach" In International Journal of Robotics Research 10(6), pp. 628-649, 1991

[Brock99] O. Brock and O. Kathib: "Elastic Strips: A framework for integrated planning and execution", in P. Corke and J. Trevelyan (Eds.), Proceedings of the International Symposium on Experimental Robotics, Volume 250 of Lecture Notes in Control and Information Sciences, pp. 328-338, 1999, Springer Verlag

[Geraerts04] R. Geraerts and M.H. Overmars: "Clearance based path optimization for motion planning", in Proceedings of the International Conference on Robotics and Automation, pp. 2386-2392, Vol. 3, April 26 – May 1, 2004, New Orleans, USA

[Gecks07] T.Gecks and D. Henrich: "Path Planning and Execution in Fast-Changing Environments with Known and Unknown Objects", In Proceedings of the International Conference on Intelligent Robots and Systems, pp. 21-26 San Diego, USA, 2007

[Henrich98] D. Henrich, H. Wörn and C. Wurll: „On-line path planning with optimal C-space discretization", in Proceedings of the International Conference on Intelligent Robots and Systems, Victoria, Canada, October 12-16, 1998

[Holleman00] C. Holleman, L. E. Kavraki: „A Framework for Using the Workspace Medial Axis in PRM Planners", In Proceedings of the International Conference on Robotics and Automation, pp. 1408-1413, April 24-28, 2000, San Francisco, USA

[Khatib86] O. Khatib: "Real-time obstacle avoidance for manipulators and mobile robots", in International Journal of Robotics Research Vol.5 (1), pp. 90-98, 1986

[Lindemann05] S. R. Lindemann and S. M. LaValle: "Smoothly blending vector fields for global robot navigation", In Proceedings of the 44th IEEE Conference on Decision & Control, pp. 3353-3559, December 12-15, 2005, Sevilla, Spain

[Likhachev05] M. Likhachev, D. Fergusson, G. Gordon, A. Stentz and S. Thrun: "Anytime dynamic a*: An anytime, replanning algorithm", In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2005

[Quinlan93] S. Quinlan and O. Kathib: "Elastic bands: Connecting path planning and control", In Proceedings of the International Conference on Robotics and Automation, Vol.2, pp802-807, 1993

[Schwartz83] J.T. Schwartz and M. Sharir: „On the piano movers problem: II. General techniques for computing topological properties of real algebraic manifolds", In Advances in Applied Mathematics 1(4), pp. 293-351, 1983