IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'10) Oct. 2010

Real-Time Path Planning for a Robot Arm in Changing Environments

Tobias Kunz, Ulrich Reiser, Mike Stilman and Alexander Verl

***Abstract*— We present a practical strategy for real-time path planning for articulated robot arms in changing environments by integrating PRM for Changing Environments with 3D sensor data. Our implementation on Care-O-Bot 3 identifies bottlenecks in the algorithm and introduces new methods that solve the overall task of detecting obstacles and planning a path around them in under 100 ms.**

A fast planner is necessary to enable the robot to react to quickly changing human environments. We have tested our implementation in real-world experiments where a human subject enters the manipulation area, is detected and safely avoided by the robot. This capability is critical for future applications in automation and service robotics where humans will work closely with robots to jointly perform tasks.

I. Introduction

Existing implementations of robot planners have successfully accomplished tasks in well-known environments like factory work cells. We study the more challenging problem of a robot acting safely and efficiently in unstructured, dynamic and human environments. Safe interaction with humans is critical for future robot applications in automation and service where humans will work closely with robots to jointly perform tasks. In order to maximize safety, robots must match and exceed the human capacity to safely respond to changes in the environment. Since human reaction time is approximately 180 ms [1], the robot should respond at least as fast. This work presents our implementation of online motion planning that achieves faster-than-human replanning.

Online planning in dynamic environments is a challenging problem because all components involved in the perception-action cycle must be efficient. We require sensors that are capable of scanning the environment in a fast and reliable manner, a representation of the sensor data that is fast to generate and suitable for the planning algorithm used, and finally a planner that is able to plan a motion in a high-dimensional space quickly while avoiding obstacles.

We present an approach that is able to detect obstacles and plan a path around them within 100 ms, nearly twice the reaction speed of a human being. Our approach is based on Dynamic Roadmaps (DRM), which were introduced by Leven and Hutchinson [2]. Like the standard PRM, DRM uses pre-computation to reduce the continuous state space to a graph for faster online search. However, while general PRM approaches focus on fixed environments, DRM incorporates data gathered to invalidate edges online and quickly respond to environment variations.

Tobias Kunz and Mike Stilman are with the Center for Robotics and Intelligent Machines (RIM) at the Georgia Institute of Technology, Atlanta, GA 30332, USA. {tobias, golem}@gatech.edu

Tobias Kunz, Ulrich Reiser and Alexander Verl are with Fraunhofer IPA, Stuttgart, Germany. {reiser, verl}@ipa.fraunhofer.de

This work was funded as part of the research project DESIRE by the German Federal Ministry of Education and Research (BMBF) under grant no. 01IME01A.

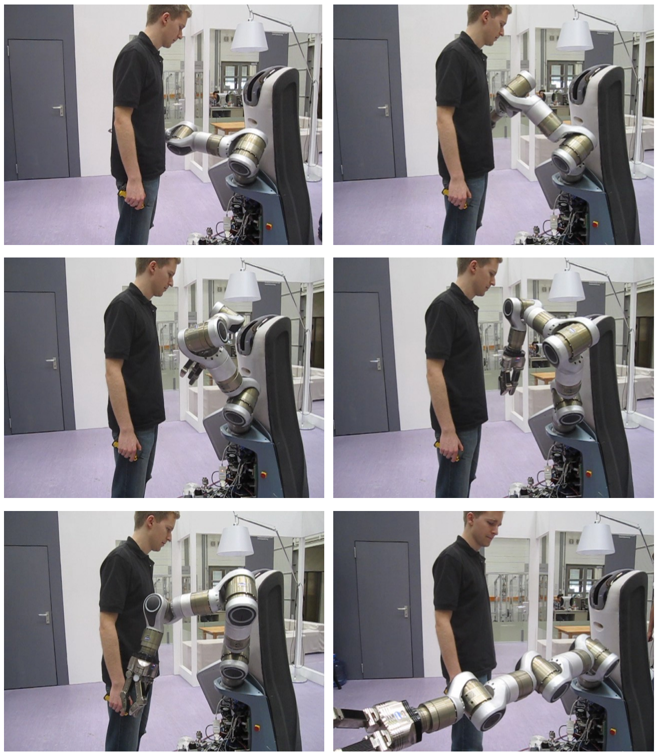

Fig. 1. Planning around a human obstacle in less than 100 ms.

In this paper, we identify bottlenecks in the algorithm and present the new techniques developed during our implementation that were critical in achieving real-time results. We have implemented our algorithm on the Care-O-bot 3 platform using a SwissRanger SR3000 time-of-flight sensor to detect obstacles. Our experiments show that our system consistently detects a human obstacle and plans a collision-free path in less than 100 ms.

The paper is organized as follows: Related work is presented in section II. The DRM algorithm is described in section III. In section IV, we describe our implementation, focussing on improvements over the original algorithm that are critical to achieving the real-time results. The results of our experiments on Care-O-bot 3 are presented in section V.

II. Related Work

Work related to robot motion strategies in dynamic environments can be divided into reactive control, global planning and mixed approaches. Reactive control approaches are mainly based on potential field methods, e.g., [3], that rely on local information. The missing global information is an inherent drawback of potential field approaches. The robot might get stuck in a local minimum and a globally optimal solution cannot be guaranteed.

Path planning approaches on the other hand take global information into account. However, they need to perform a time consuming search for a collision-free path depending on the current states of the robot and the environment. In the last decade, path planning in high-dimensional configuration spaces has been dramatically expedited through sampling based techniques, which subordinate optimality to achieve improved performance [4].

Sampling-based planners can be distinguished into single-query and multi-query planners. Single-query planners, such as RRT (Rapidly Exploring Random Trees) [5] and SBL (Single-query Bidirectional PRM Planner with Lazy Collision Checking) [6] plan from the start for each query. While they handle changing environments, exploring the configuration space and checking possible path segments for collision still requires a high computational effort. Therefore, direct application of these methods to real-time replanning is currently infeasible.

Multi-query planners like the Probabilistic Roadmap (PRM) Planner assume that several different planning queries are to be executed within the same static environment. [7] Therefore, they can spend a large amount of time on preprocessing to create a data structure that speeds up the individual planning queries. However, due to the assumption of a static environment these methods are not directly applicable to changing environments.

Some approaches, e. g. [8], make use of a geometric model of the environment. However, building a geometric model from raw sensor data is difficult and time-consuming. Since collision avoidance is possible using a more basic representation of the sensor data, such a high level representation of the environment is not necessary if its only purpose is collision avoidance.

Elastic bands [9], [10] and elastic strips [11] are mixed approaches, which combine global and local information. They assume an initial globally planned path. This path is adapted to a changing environment using local reactive control. The approach uses a geometric representation of the distance from the robot to the closest obstacle to adapt the path. However, calculating distances from the robot to a point cloud might be time consuming. We are not aware of an existing implementation that shows the performance of these approaches on a real robot using real sensor data. Furthermore, elastic bands and elastic strips are more complex than our approach and do not provide performance improvements for initial planning.

Dynamic Roadmaps (DRM) introduced by Leven and Hutchinson [2] adopt the method of increasing online efficiency through preprocessing from classical PRM approaches. They adapt the classical PRM by generating a preprocessed roadmap that can still be used in changing environments. This is achieved by invalidating the parts of the roadmap that are blocked by dynamic obstacles before every planning round. To enable real-time planning, we need to be able to quickly determine which parts of the roadmap are blocked at any given time. To achieve this, DRM relies on preprocessing again. The workspace is discretized on a grid and a precomputed mapping between the workspace grid and the roadmap is used for fast invalidation.

Leven and Hutchinson [2] report a planning time of one second for a 6-DOF robot arm in simulation using a 2048-node roadmap. We consider one second to be too slow to react fast enough to a changing human environment. More recent papers [12], [13], which adapt the approach in [2], report planning times comparable to ours. However, all existing approaches are only run in simulation using a geometric model of a simple environment. On a real robot the geometric model needs to be created from raw sensor data. This takes additional time, which needs to be taken into account to evaluate the overall real-time capabilitites.

In contrast, we do not assume access to a geometric model of the environment. Instead, we use a raw point cloud, which makes collision checking slower. In the following sections we show improvements to the existing implementations to enable equally fast or faster planning, although we are using real sensor data. Unlike [12] and [13], our focus is on avoiding collision checking as much as possible. Because of that our implementation is able to plan in less than 100 ms on a real robot using real sensor data.

III. Dynamic Roadmaps

This section explains the baseline DRM as introduced by Leven and Hutchinson [2].

DRM uses a precomputed workspace mapping for fast invalidation of blocked roadmap parts. Each workspace grid cell stores a list of roadmap nodes and edges that are in collision with the cell. During the preprocessing stage for each node and edge of the roadmap all workspace grid cells that are in collision with a given configuration or set of configurations are computed and the roadmap node or edge is stored in the collision list of each respective grid cell. This discretization may lead to a larger part of the roadmap being invalidated than actually is in collision with the obstacles.

Since we build the roadmap offline, we are not bounded by time in assuring that we achieve a good roadmap. The quality of the roadmap depends on the distance metric used. As we do not have to calculate the distances online, we can use a better, more time-consuming metric. Leven and Hutchinson [2] propose the following two metrics among others. Both metrics capture the motion of the robot in workspace. They use a set of reference points on the robot surface and base the distance on how these reference points move in workspace.

The first metric is solely based on the locations of the reference points at the two end configurations. It calculates the distance of each reference point between the two configurations and takes the L^2 norm of the vector consisting of all the distances. This *direct workspace metric* is defined as

$$d_2^{\mathcal{W}}(p,q) = \sqrt{\sum_{a \in \mathcal{A}} \|a(p) - a(q)\|^2} \tag{1}$$

where $\mathcal{A}$ is the set of all reference points on the robot surface and $a(p)$ is the location of reference point a in workspace given the robot is in configuration p.

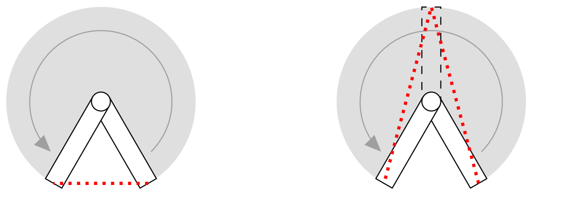

Fig. 2. Example for the midpoint improving the metric.

Eq. 1 does not consider the motion of the robot in-between the two end configurations. It is possible that the two end configurations are close in workspace but the robot sweeps a large volume while transitioning between them. Fig. 2 shows this situation for a one-link robot. The robot transitions between two configurations sweeping the gray area. Although the swept area is large, the workspace distance (shown with a red dotted line) used by the direct workspace metric is small. To counter this problem, the second metric incorporates a midpoint $m = \frac{p+q}{2}$. The motion between the two end configurations is partially captured by Eq. 2.

$$d_{2m}^{\mathcal{W}}(p,q) = \sqrt{(d_2^{\mathcal{W}}(p,m))^2 + (d_2^{\mathcal{W}}(m,q))^2} \tag{2}$$

This *midpoint workspace metric* is important for achieving a high-quality roadmap. This is why we use it in our implementation for calculating the path cost. However, this is only a pseudo-metric, because it does not satisfy the triangle inequality, and the inclusion of the midpoint increases computation time. Therefore, in our implementation we also use other metrics that build on top of the direct workspace metric defined in (1) in order to achieve a reasonable trade-off between quality and time efficiency in every situation.

IV. Implementation

DRM assumes we know which workspace grid cells are blocked and leaves it to a particular implementation to choose how to determine them. We use a SwissRanger SR3000 time-of-flight sensor. The overall system design that incorporates the sensor and DRM is sketched in Fig. 3. The system is implemented on Care-O-bot 3 [14] which integrates a 7-DOF manipulator and a 3D time-of-flight sensor.

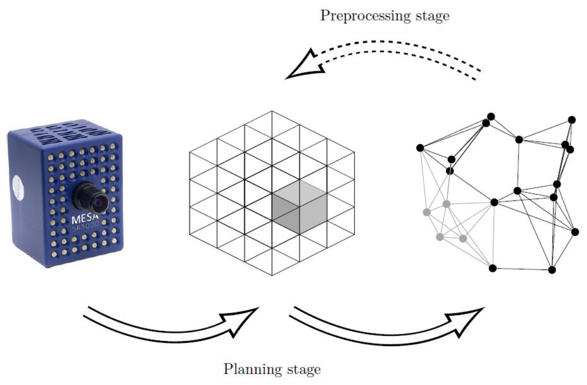

Fig. 3. Design schematic: sensor, workspace grid and roadmap

Our implementation is based on the DRM algorithm. However, the real-time capabilities that we report are specific to our efficient implementation. The two critical advances responsible for achieving real-time results are as follows:

- Efficient connection of the start and goal configurations to the roadmap as described in section IV-B.2
- A* search using an admissible, consistent and fast heuristic as described in section IV-B.3.

In the remainder of this section we describe the details of our implementation for each step of the DRM algorithm. We present the methods used in preprocessing, followed by the algorithms in online planning.

A. Preprocessing Stage

1) Roadmap Generation: The nodes of the roadmap are created by sampling the 7-dimensional configuration space of the robot uniformly. Each node is connected to its k nearest neighbors provided that the segment is collision-free. The metric used is the midpoint workspace metric defined in (2).

The collision detection algorithm of the Care-O-bot manipulation framework [15] is used to check discrete configurations at roadmap nodes and along edges for self-collision. As this collision checker does not return any information on the distance to the closest obstacle, we cannot guarantee that a roadmap edge is collision-free. However, we can trade off the likelihood of not detecting a collision and the time needed for collision checking by choosing a upper bound on the distance ϵ between two checked configurations. In order for ϵ to be a good measure of how likely it is to miss a collision, we base ϵ on a metric similar to the direct workspace metric defined in (1). But instead of taking the L^2 norm of the distance vector, we take the L^∞ norm, resulting in

$$d_\infty^{\mathcal{W}}(p,q) = \max_{a \in \mathcal{A}} \|a(p) - a(q)\| \tag{3}$$

This gives us the maximum amount that any reference point is displaced between two checked configurations. If the robot is bounded by convex polyhedrons and all their vertices are chosen as reference points, the metric gives the maximum displacement of any point on the robot. [4]

2) Workspace Mapping: To allow for fast invalidation of blocked roadmap parts, DRM creates a mapping between the workspace grid and the configuration space roadmap. The grid consists of hypercube cells with a constant but configurable edge length, covering the whole workspace of the arm. To create this mapping, we need to determine for each grid cell at which nodes and edges of the roadmap the robot occupies the cell. The mapping is created backwards by determining all grid cells that are occupied in a particular configuration. This process is also referred to as voxelization. To achieve this we chose a suboptimal and slow but very simple method. We uniformly distribute about 4,500 points over the volume and the surface of the manipulator and map them into the workspace grid. Every grid cell that is hit by at least one of these points is occupied by the robot at a given configuration. For determining the set of grid cells that are swept while moving along a roadmap edge, we use the same method as Leven and Hutchinson [2]. First, the grid cells corresponding to the midpoint of the path segment are added to the set. The path segment is then recursively subdivided into two parts and the process is repeated until no new occupied cells are added to the set.

B. Planning Stage

1) Invalidating Blocked Parts of the Roadmap: The first step in executing online planning is the determination of blocked parts of the roadmap. Since we have precomputed a mapping from the workspace grid to the roadmap, we only need to determine the blocked workspace cells. Obstacles are detected using the SwissRanger SR3000 distance sensor which provides a point cloud in the coordinate frame of the sensor. Each point is transformed into the robot base coordinate frame and then discretized. If it hits a grid cell, all roadmap nodes and edges that are in collision with the grid cell are invalidated for the current planning round. This invalidation can be performed efficiently since the algorithm iterates through two lists of pointers to nodes and edges, marking invalid entries. After a grid cell has been hit once, it is marked as blocked. Hence, future hits perform no further computation marking previously invalidated roadmap edges.

2) Connecting Start and Goal Configurations to the Roadmap: In order to plan a path from the start to the goal configuration, we need to connect both the start and goal configurations to the roadmap. The connections must be collision-free. This is the only time we have to perform collision checks during online planning.

Connecting the start and goal configurations to the roadmap requires the detection of k nearest neighbors for each configuration. We use a cover-tree data structure [16], which permits fast nearest neighbor searches in arbitrary metric spaces. Although we emphasized the midpoint workspace metric defined in (2) for all other distance calculations, this step requires the metric presented in (4).

While the midpoint metric is more informed, it has significant overhead because it computes forward kinematics for all three reference points: start, middle and end. Furthermore, it does not satisfy the triangle inequality, which data structures for fast nearest neighbor searches like the cover-tree rely on. While any workspace metric must compute the forward kinematics for the terminal states (start and end), we choose one that does not require the midpoint. This is very important since the midpoint between a terminal state and every state in the graph will change depending on the location of the terminal state. Hence, using the midpoint metric here would require the online planner to compute forward kinematics for *twice as many states as there are nodes in the graph.*

Instead of the midpoint metric, we use a combination of the direct workspace metric shown in (1) and a weighted L^1 metric in configuration space. The direct workspace metric still requires the calculation of the locations of reference points for the terminal configurations. However, to speed up the metric, we can precompute the reference points for all roadmap nodes during the preprocessing stage and store them with the nodes. Thus, the metric only requires two computations of forward kinematics online.

$$d_{\text{combined}}(p,q) = 0.9 \cdot d_2^{\mathcal{W}}(p,q) + 0.1 \cdot d_{w1}(p,q) \tag{4}$$
$$d_{w1}(p,q) = \sum_{i=1}^{n} w_i |p_i - q_i|$$

The k nearest neighbors are considered as candidates for connection and the generated edges are provided as input to the shortest path algorithm. In the spirit of SBL [6] and unlike [2], collision checking of these edges is delayed. However, unlike SBL, we don't delay collision checking until a candidate path has been found but only until an unchecked edge becomes part of the closed set (i.e. when the second node of the edge is expanded) during graph search. This is to make sure the search does not have to backtrack after an edge has been found to be in collision.

3) Graph Search: After the blocked parts of the roadmap have been invalidated and the start and goal configurations have been connected to the roadmap, the actual search for the shortest collision-free path is conducted. Unlike [2] we use the A* algorithm [17]. The heuristic used is the distance to the goal with the following metric, which is the direct workspace metric multiplied by a constant factor.

$$d_{2c}^{\mathcal{W}}(p,q) = \sqrt{\frac{1}{2}}\, d_2^{\mathcal{W}}(p,q) = \sqrt{\frac{1}{2}\sum_{a\in\mathcal{A}} \|a(p) - a(q)\|^2} \tag{5}$$

This heuristic is admissible and consistent ($h(x) \leq c(x,y) + h(y)$) because the metric satisfies the triangle inequality and is a lower bound on the midpoint workspace metric, which is used for the cost function. The following inequality proves the fact that the metric used for the heuristic is a lower bound on the midpoint workspace metric.

$$\begin{aligned} d_{2m}^{\mathcal{W}}(p,q) &= \sqrt{\sum_{a\in\mathcal{A}} \|a(p)-a(m)\|^2 + \sum_{a\in\mathcal{A}} \|a(m)-a(q)\|^2} \\ &\geq \sqrt{\sum_{a\in\mathcal{A}} \left\|a(p)-\frac{a(p)+a(q)}{2}\right\|^2 + \sum_{a\in\mathcal{A}} \left\|\frac{a(p)+a(q)}{2}-a(q)\right\|^2} \\ &= \sqrt{\frac{1}{2}\sum_{a\in\mathcal{A}} \|a(p)-a(q)\|^2} = d_{2c}^{\mathcal{W}}(p,q) \end{aligned} \tag{6}$$

In order for the A* search to yield a speed-up compared to an uninformed search, the heuristic needs to be calculated quickly. In our case this was only possible because of the precalculation of the reference points described in the previous section.

V. Experimental Results

In this section we present results from experiments run on the Care-O-bot 3 plattfrom. Our experiments show that our implementation of DRM is able to plan a path around sensor-detected obstacles in less than 100 ms. We show that the details of our implementation are crucial in achieving the real-time requirements.

The algorithm has several parameters that can be tuned. Table I shows these parameters and the values we used for the experiments unless otherwise stated.

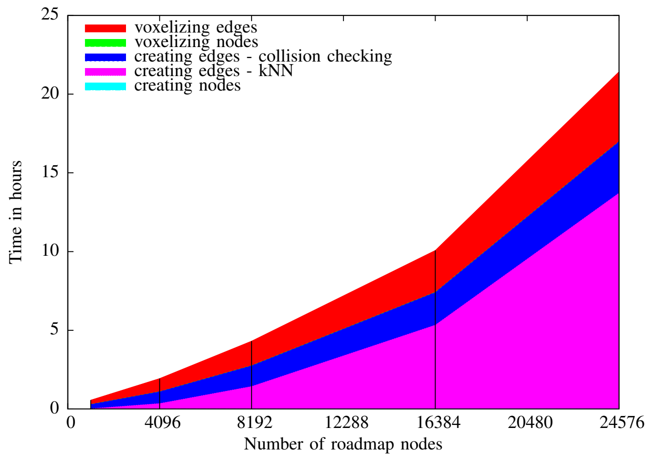

Fig. 4. Preprocessing time

Symbol	Value	Description
n	16384	Number of roadmap nodes
k	20	The number of nearest neighbors that are considered for creating an edge to.
$\epsilon_{\text{roadmap}}$	0.01 m	Maximum distance between two collision-checked configurations on a path segment while creating the roadmap.
$\epsilon_{\text{realtime}}$	0.05 m	Same as above. But while connecting the start and goal configurations during the planning stage, we use a larger ϵ because time is more valuable.
l_{grid}	0.05 m	Edge length of a cubic workspace grid cell.

TABLE I

CONFIGURABLE PARAMETERS AND THEIR DEFAULT VALUES.

A. Preprocessing Stage

Timing results for the preprocessing stage were retrieved on a 3 GHz Intel Pentium 4.

As Fig. 4 shows, the preprocessing takes several hours and depends on the size of the roadmap. Time needed to generate and voxelize nodes is negligible compared to the other subtasks. Searching for the k nearest neighbors of each of the n nodes takes by far the most time. This is because we use a brute-force search calculating n^2 distances. We did not use a more sophisticated algorithm for nearest neighbor search, e.g. a cover-tree, because it cannot guarantee an exact result for the pseudometric used here and time is not very critical during the preprocessing stage.

With 16,384 nodes our roadmap contained about 188,000 edges. I. e. in average each node has 23 neighbors. The resulting size of the generated roadmap including the workspace mapping depends on the number of nodes and on the grid resolution. Our default roadmap with 16,384 nodes and a grid resolution of 0.05 m results in a file size of about 250 MB when written to disk. The roadmap size in main memory is a little bit less than that. For the same grid resolution, but 24,576 nodes, the file size was 325 MB. For the default number of nodes but twice the workspace grid resolution, i.e. 0.025 m, the file size increased to 1.8 GB.

B. Planning Stage

Timing results for the planning stage were retrieved on a 2 GHz Intel Pentium M.

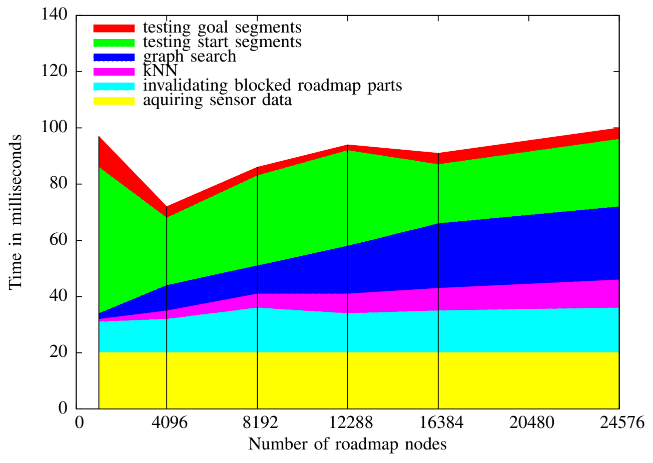

Fig. 5. Planning time

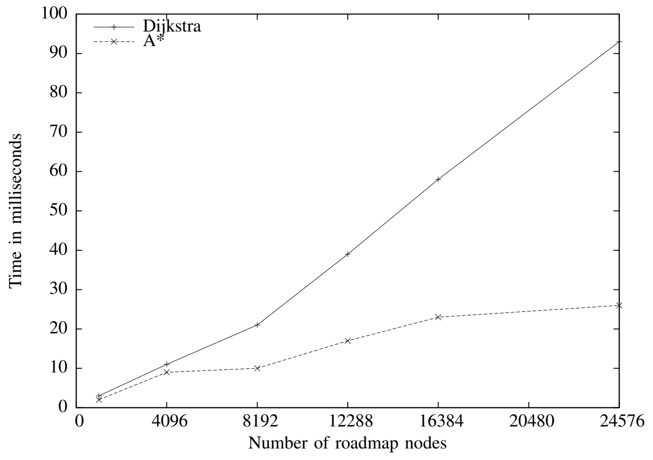

Fig. 6. Comparison of search times using A* and Dijkstra's algorithm

Fig. 5 shows the time required for one online planning step subdivided by subtasks. The figure shows the main result of our paper: The total planning time needed is less than 100 ms, almost independent of the number of roadmap nodes. The kNN and graph searches take more time with growing roadmap size. Testing path segments, in contrast, takes less time with growing roadmap size. This is because a larger roadmap means the configuration space is more densely covered by the roadmap. Thus, the k nearest neighbors of the start and goal configurations are closer to the start and goal configurations and fewer collision checks are necessary to test those path segments. Acquiring the sensor data over the network currently takes about 20 ms, which is quite long and could probably be optimized.

One reason for the fact that our algorithm scales well with roadmap size is the use of A* for the graph search. Fig. 6 compares the runtime of A* with an uninformed search using Dijkstra's algorithm, which might have been used in [2]. As we will see in the next paragraph, A* also helps avoiding collision checks.

Testing goal segments takes less time than testing start segments because of our delayed collision checking of these segments. Because we use a heuristic-guided A* search, not all of the edges connecting the start configuration become

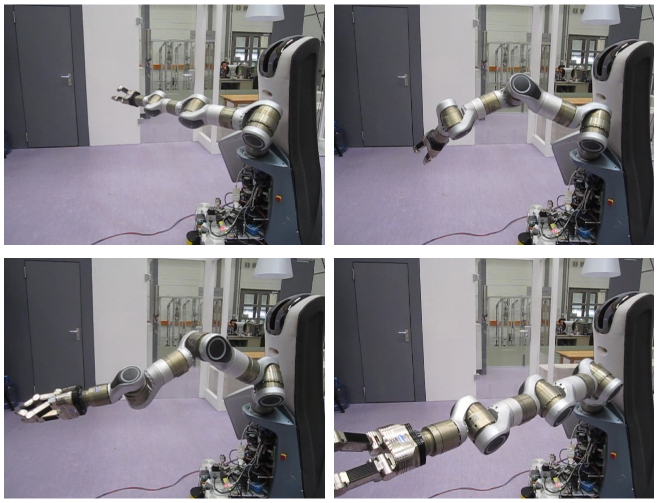

Fig. 7. Planning without obstacles.

part of the closed set. Thus, not all of these edges need to be checked for collision. In the experiments shown in Fig. 5 between 10 and 16 out of the 20 start edges are checked for collision. Out of the edges connecting the goal configuration, even fewer need to be checked for collision because the search terminates after having found the first collision-free of these edges. In the experiments shown in Fig. 5 always only one edge is checked for collision. This shows an improvement over the original approach in [2].

C. Resulting Motion

Fig. 1 and 7 show the paths produced by the planner on the real robot for two different scenarios. In both cases the robot is moving in between the same two configurations. In Fig. 7 there is no obstacle blocking the motion. In Fig. 1 a human is blocking the direct path. The robot quickly finds a path around the obstacle. Because we restrict the motion of the robot to a pregenerated roadmap, we might not find a solution in a cluttered environment although one exists. However, our experiment pictured in Fig. 1 shows that this is not a problem in practice. The robot is able to find a solution although the free space is narrow.

VI. Conclusion

In this work we presented an integrated solution for real-time planning in changing environments using 3D sensor data. Our path planner was based on the DRM algorithm [2] and was implemented on the mobile robot platform Care-O-bot 3 [14]. Our specific implementation was critical to achieving the real-time results. We presented a heuristic that allowed for optimal A* search. We enabled a fast connection of the start and goal configurations to the roadmap by delaying collision checking and by a fast search of the k nearest neighbors by using an appropriate metric.

Our experiments show that our integrated solution is able to calculate a collision-free path for the 7-DOF manipulator of Care-O-bot 3 within less than 100 milliseconds. This includes the whole perception-action cycle from acquiring 3D sensor data and maintaining an obstacle model to the shortest collision-free path search. Thus, we are able to react quickly to a changing human environment.

Although an efficient implementation was critical to achieving the real-time results, there are many ways to further improve our implementation. For example, a hierarchical workspace grid with different cell sizes on different levels would lead to a smaller workspace mapping, a faster invalidation of blocked roadmap parts and would allow a more fine-grained workspace discretization. Also, the determination of which workspace cells are occupied by the robot in a given configuration may be improved by using an algorithm that directly voxelizes polyhedra, e. g. [18]. This would speed-up collision checking of the start and goal segments and the generation of the workspace mapping during the preprocessing stage.

References

[1] J. M. T. Brebner and A. T. Welford, “Introduction: An historical background sketch,” in *Reaction Times*, A. T. Welford, Ed. Academic Press, 1980, pp. 1–23.

[2] P. Leven and S. Hutchinson, “A Framework for Real-time Path Planning in Changing Environments,” *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 999–1030, 2002.

[3] O. Khatib, “Real-Time Obstacle Avoidance for Manipulators and Mobile Robots,” *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[4] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[5] S. LaValle and J. Kuffner, “Rapidly-exploring random trees: Progress and prospects,” in *Algorithmic and computational robotics: new directions: the fourth Workshop on the Algorithmic Foundations of Robotics*. AK Peters, Ltd., 2001, p. 293.

[6] G. Sanchez and J.-C. Latombe, “A single-query bi-directional probabilistic roadmap planner with lazy collision checking,” in *Int. Symp. Robotics Research*, 2001, pp. 403–417.

[7] L. Kavraki and J. Latombe, “Probabilistic roadmaps for robot path planning,” *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, vol. 53, 1998.

[8] Y. Yang and O. Brock, “Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation in dynamic environments,” in *Proc. of Robotics: Science and Systems*, August 2006.

[9] S. Quinlan and O. Khatib, “Elastic bands: connecting path planning and control,” *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, pp. 802–807 vol.2, May 1993.

[10] S. Quinlan, “Real-time modification of collision-free paths,” Ph.D. dissertation, Stanford University, Stanford, CA, USA, 1995.

[11] O. Brock and O. Khatib, “Elastic Strips: A Framework for Motion Generation in Human Environments,” *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, 2002.

[12] M. Kallman and M. Mataric, “Motion planning using dynamic roadmaps,” in *2004 IEEE International Conference on Robotics and Automation*, vol. 5, 26 2004.

[13] H. Liu, X. Deng, H. Zha, and D. Ding, “A path planner in changing environments by using w-c nodes mapping coupled with lazy edges evaluation,” in *Int. Conf. on Intelligent Robots and Systems*, 2006.

[14] U. Reiser, C. Connette, J. Fischer, J. Kubacki, A. Bubeck, F. Weisshardt, T. Jacobs, C. Parlitz, M. Hägele, and A. Verl, “Care-o-bot 3 - creating a product vision for service robot applications by integrating design and technology.” in *IROS*. IEEE, 2009.

[15] U. Reiser, R. Volz, and F. Geibel, “ManIPA: a flexible manipulation framework for collision avoidance and robot control.” *39th International Symposium on Robotics*, pp. 407–411, Oct. 2008.

[16] A. Beygelzimer, S. Kakade, and J. Langford, “Cover trees for nearest neighbor,” in *ICML ’06: Proc. of the 23rd international conference on Machine learning*. New York, NY, USA: ACM, 2006.

[17] P. Hart, N. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, July 1968.

[18] A. Kaufman and E. Shimony, “3d scan-conversion algorithms for voxel-based graphics,” in *SI3D ’86: Proceedings of the 1986 workshop on Interactive 3D graphics*.