6 DOF path planning in dynamic environments – A parallel on-line approach

Dominik HENRICH, Christian WURLL and Heinz WÖRN

Institute for Process Control and Robotics (IPR)
Computer Science Department, University of Karlsruhe
Kaiserstrasse 12, D-76128 Karlsruhe, Germany
e-mail: dHenrich@ira.uka.de, http: //wwwipr.ira.uka.de/~paro/

Abstract

This paper presents a new approach to parallel path planning for industrial robot arms with six degrees of freedom in an on-line given 3D environment. The method is based a best-first search algorithm and needs no essential off-line computations. The algorithm works in an implicitly discrete configuration space. Collisions are detected in the Cartesian workspace by hierarchical distance computation based on polyhedral models of the robot and the obstacles. By decomposing the 6D configuration space into hypercubes and cyclically mapping them onto multiple processing units, a good load distribution can be achieved. We have implemented the parallel path planner on a workstation cluster with 9 PCs and tested the planner for several benchmark environments. With optimal discretisation, the new approach usually shows very good speedups. In on-line provided environments with static obstacles, the parallel planning times are only a few seconds.

1 Introduction

The issue of robot path planning has been studied for a couple of decades and many important contributions to the problem have been made [9]. Path planning algorithms are of great theoretical interest, but are rarely used in practice because of their computational complexity [10]. Here, we make a step in the direction of practical path planning.

Many of the future robotic tasks (e.g. recycling, robot guidance, tele-operation, assembly and disassembly, medical surgery) can often only be completed in dynamic environments. Therefore, powerful on-line path planners for industrial robots with six degrees of freedom (DOF) are needed. The *on-line* capability[1] means that the planner does not require any time-consuming off-line computations in order to directly react to dynamic changes in the environment.

For dynamic environments, several different cases can be distinguished. In the first case, the environment contains dynamic obstacles (e.g. objects on a conveyor belt, or additional robots) with known or partially known movements. In an other case, the robot grips different objects. There, the kinematic chain of the robot, including the gripped object, will change. The next case occurs in the area of virtual engineering. After every assembly operation, the product or the environment will change its geometry. All these cases of dynamic environments implicate a modification of the configuration space (C-space), which has to be considered during planning paths.

An introduction to motion planning in dynamic environments is given in [3]. In several examples, different approaches especially for mobile robots are presented. In [2], a motion planner for industrial robots based on velocity adaptation is discussed. It plans only for a 2 DOF workspace for two robots and their off-line known movements. In [14], a potential-field approach based on the explicit calculation of the workspace and the C-space is proposed. When a new object is detected, the new path is sought within a few seconds, but the planner works only with 5 DOF in a very small search space, which is unfavorable for industrial applications.

Generally speaking, speeding up the computation will enable the path planner to cope better with dynamic environments in practice. One approach is based on the introduction of parallel processing. Unfortunately, these approaches to parallel path planning are restricted either to simple geometric object representations (coarse grids or line segments), to a reduced number of DOF, or to only static environments were a C-space representation can be used in a simplified way [7].

In summary, to date, no planners for 6 DOF robots exist, which can deal with dynamic environments and have low on-line computation times. Our aim is to develop a path planner satisfying these requirements for robots with up to 6 DOF [17]. We focus on industrial robots, which constitute a considerable fraction of all robots used currently and in future.

The remainder of the paper is organized as follows: Section 2 introduces the basic approach of our path planner. Section 3 describes the necessary enhancements for paral-

[1] Here, "on-line" does not include to meet given time constrains as required for "real-time".

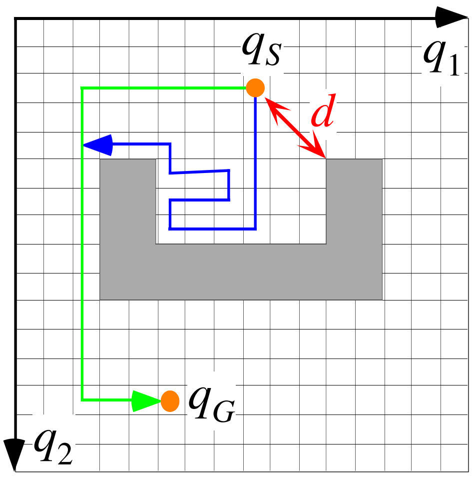

Figure 1: A 2D illustration of the path search in the implicit C-space from the start q_S to the goal q_G using the Cartesian obstacle distance d for collision detection

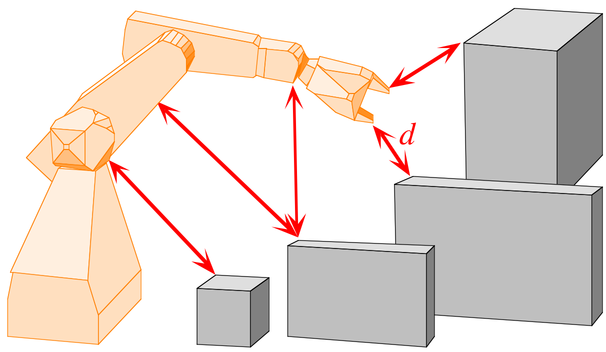

Figure 2: Collision detection in the explicit workspace by computing the minimum distance d between robot and obstacles

lelizing the sequential approach. Section 4 shows the experimental results for load balancing, message combining, and speedups. Section 5 gives an conclusion and an outlook to the future investigations.

2 Sequential approach

Most of the off-line path planners are based on an explicit representation of the *free C-space*. The free C-space computation consists of the obstacle transformation into the C-space and the construction of a free-space representation. Both tasks are very time- and memory consuming, and their calculation effort increases with the robot's DOF. In order to avoid these time consuming obstacle transformations, one can search in an implicitly represented C-space and detect collisions in the Cartesian workspace.[2] This strategy enables the planner to cope with on-line provided environments and moving obstacles. See Figure 1 and Figure 2.

For searching in the implicit C-space, any best-first search mechanism can be applied. We choose a variation of the well known A*-search algorithm [4]. Robot configurations (nodes) still to be processed are stored in OPEN, while already processed nodes are stored in CLOSED. Contrasting to the original A*, here, no re-opening of nodes in CLOSED is performed. As evaluation function $f(n) = (1-w)\,g(n) + wh(n)$ is used, where $g(n)$ is the number of nodes of the path from the start node to node n, and $h(n)$ is the Airplane distance in C-space between node n to the goal node. Increasing the weight $w \in [0, 1]$ beyond 0.5 generally decreases the number of investigated nodes while increasing the cost of the solutions generated. To improve the on-line capabilities of the path planner, our search is strongly directed to the goal by setting $w = 0.99$ [15].

Collisions are detected by a fast, hierarchical distance computation in the 3D workspace, based on the polyhedral model of the environment and the robot provided by common CAD systems [5, 6]. With the help of the "MaxMove Tables", introduced in [11], the Cartesian distances are then transformed into joint angles in order to determine whether the current configuration collides or not. For obtaining similar joint intervals, thus implicating an efficient distance exploitation, the optimal joint discretisation is automatically computed based on the method of [13].

3 Parallel approach

For parallelizing the A*-algorithm, the configurations in OPEN and CLOSED must be accessible to all processors in order to distribute them. These lists can either be managed by one dedicated processor or each processor can have its own local lists. In a message passing system, each access to a global list would amount to an enormous communication effort. Thus, the local method was preferred.

The work distribution is the key aspect of parallelization. Therefore, the C-space is decomposed into d-dimensional hypercubes of size b in each dimension. For parallel processing, the hypercubes are cyclically mapped

[2] Verwer is one of the first, who has used this strategy [16]. The reported planning times are acceptable for 2 DOF examples but not for 5 DOF in order to cope with dynamic environments.

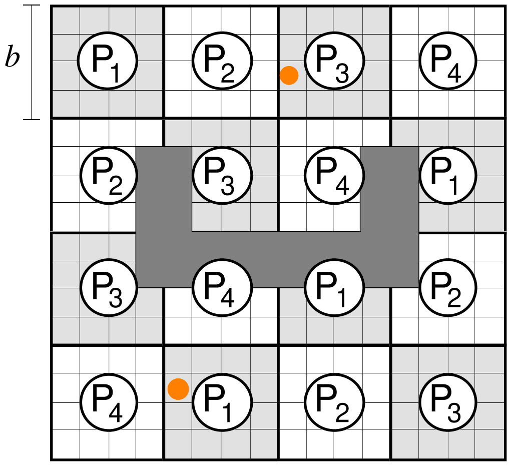

Figure 3: 2D illustration of the C-space decomposition in hypercubes of size b and the cyclically mapping of the hypercubes onto four processing units P_1, …, P_4.

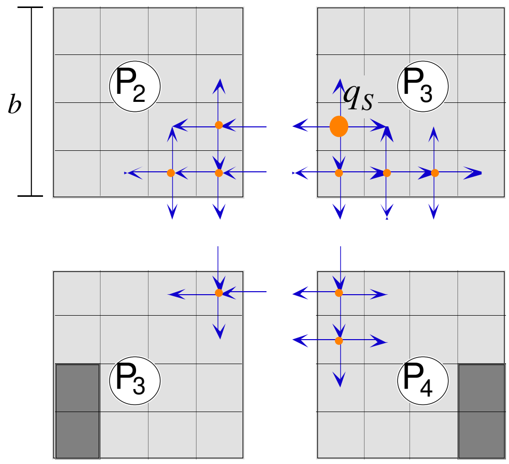

Figure 4: 2D illustration of the parallel search in four neighboring hypercubes taken from the C-space decomposition in Figure 3 starting at q_S.

onto the p available processors by the following function[3]:

$$M(q) := 1 + \left(\sum_{i=1}^{d} \left\lfloor \frac{q_i}{\Delta q_i * b} \right\rfloor \right) \bmod p$$

According to the automatically computed discretisation Δq_i, every configuration $q = (q_1, \ldots, q_d)$ is mapped uniquely to one hypercube or to one processor. Thus, the OPEN list of each processor contains configurations of the multiply mapped hypercubes.

Figure 3 demonstrates the decomposition of a 2D C-space for $p = 4$ processors and the cube size $b = 4$. The 2D cubes are mapped cyclically onto the four processors according to $M(q)$. Figure 4 shows an enlargement of four neighboring hypercubes. Each processor runs a local A*-search beginning with the hypercube containing q_S. After the search has reached the hypercube boundaries, the expanded successors are sent to their corresponding processors. The received configurations will then be inserted in a local OPEN list. Similar to the sequential version, in each iteration, every processor expands the best configuration of OPEN until the list is empty or a goal node is chosen for expansion. In the former case, if the OPEN lists of all processors are empty, the algorithm reports that there is no solution. In the latter case, the solution path is retraced across the hypercubes involved.

4 Experimental results

We have implemented the parallel path planner on a workstation cluster. The cluster consists of 9 PCs, each with 133 Mhz Intel Pentium processors and 64 Mbyte memory. The parallel communication is established by an Ethernet based bus network. For more details see [18].

For testing the path planner, we have developed five benchmark problems for a 6 DOF robot, a Puma260. As an example, the benchmark problems STAR and DETOUR developed in [11] are shown in Figure 6 and Figure 7, respectively.[4]

4.1 Load balancing

The performance of the parallel algorithm essentially depends on the load balancing mechanism. In our approach, we have implemented a static distribution mechanism, which can be influenced by modifying the cube size b. Considering the C-space decomposition, small sizes result in more cubes (in different areas of the C-space) being mapped onto a single processor. Thus implicating a good load distribution. In contrast, larger sizes make the load balancing worse. This is mainly due to the coarse decomposition. Thus, the processors are idle for a longer time before they receive work. Additionally, larger cube sizes lead to less cubes being mapped onto one processor.

For validating the performance of this load balancing mechanism, we have solved benchmark STAR with different cube sizes b. On each of the $p = 8$ processors, we measured the number of collision detections, since collision detection is the most expensive function. Figure 5 shows the experimental results. For larger values, the coarse C-space decomposition leads to an irregular load distribution. In some cases, only a few cubes cover the complete solution space. Thus, some processors become

[3] The operator $\lfloor x \rfloor$ denotes the next lower integer number of x.

[4] The data of these benchmark problems can be downloaded from the Web page at http://wwwipr.ira.uka.de/~paro/skalp/

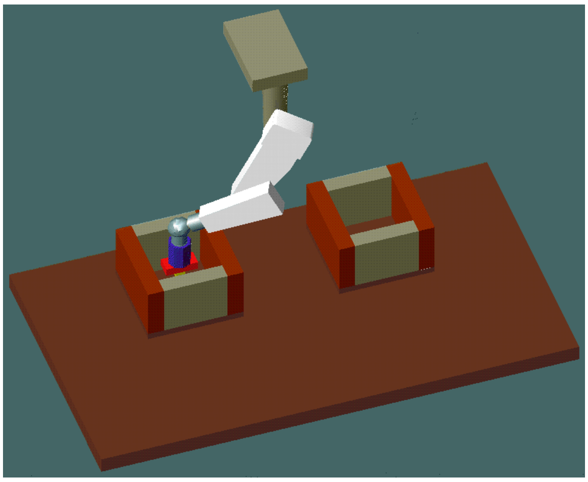

Figure 6: The benchmark problem STAR for the 6 DOF robot Puma260

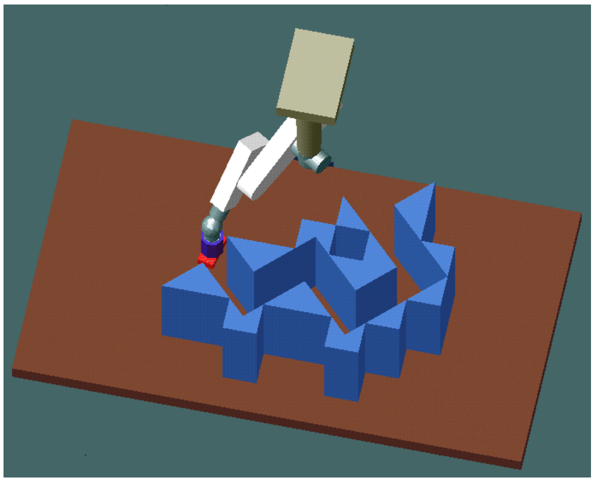

Figure 7: The benchmark problem DETOUR for the 6 DOF robot Puma260

idle. Additionally, due to the sparse mapping, the searching processors expand unnecessary configurations, since they receive no better ones. For smaller cube sizes, the load is nearly equally distributed. The corresponding runtime results for benchmarks STAR and DETOUR are depict in Figure 8.

Altogether, these results indicate that the cyclically mapping used for the parallel version (Section 3) is an efficient work distribution mechanism for this application. For the rest of the experiments we heuristically choose $b = 16$ as a good cube size in average.

4.2 Message combining

Smaller cube sizes result in a good load distribution, but increase the number of messages. Too many messages, however, usually lead to a bottleneck in the communication network and slow down the calculation times. Combining several messages to form one message seems to be a way out of this problem by reducing the number

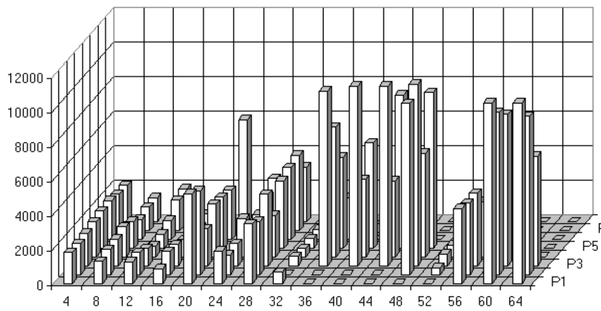

Figure 5: Number of distance computations as a measure for the processor load depending on the cube size *b* for the processors P_1, …, P_8

of message set-ups. But this holds not true as it can be seen in the experimental results.

Evaluation runs for the benchmark problems STAR and DETOUR are given in Figure 9. Here, $p = 8$ processors and a cube size of $b = 2$ are used (resulting in a rather high number of messages). The results show that the combining the configurations into one message over several iterations leads to longer planning times. For example, without combining, the benchmark problem STAR was solved in 8 sec, but with the combining of messages over 20 iterations the planner needed 20 sec [15].

This is mainly due to the fact that the remote configurations are sent too late. Thus, every processor does not receive better a configuration and so expands the worse ones. At this stage, no message combining was included in the further runtime tests.

4.3 Run-time and speedups

To compare the run-times, we have run every benchmark problem 12 times, deleting the lowest and highest planning times and computing the average of the remaining 10 values. The Cartesian resolution was chosen to be 20 *mm*, which leads (for a Puma260) to the discretisation $\Delta q = (1.91^\circ, 1.96^\circ, 2.79^\circ, 5.66^\circ, 5.66^\circ, 20.66^\circ)$. According to the upper and lower joint limits of the Puma260, the C-space consists of $2.99 * 10^{11}$ states, which is at least 3 magnitudes greater than in the reference examples known by the authors.

To calculate the speedups, we ran 8 parallel processes on 1, 2, 4, and 8 processors, thus, guaranteeing the same C-space decomposition. This method of measuring was necessary in order to obtain a fair comparison, because the

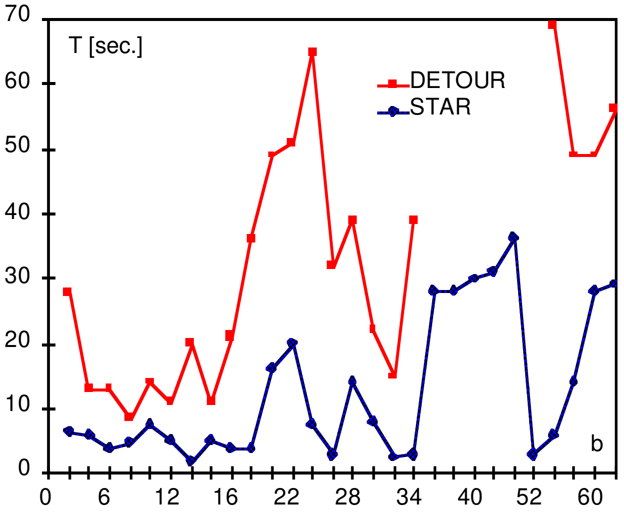

Figure 8: Run-time *T* of the parallel path planning system for increasing cube sizes *b*

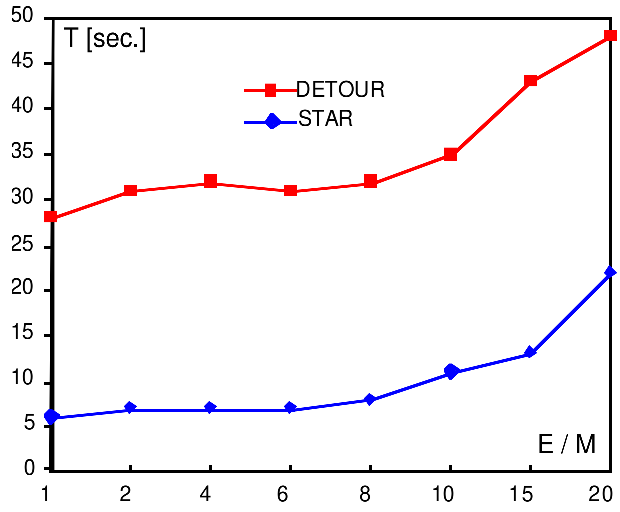

Figure 9: Run-time *T* of the parallel path planning system for different number of expansions *E* per message *M*

search performance essentially depends on the C-space decomposition (see Section 3).

The parallel planning times and the achieved speedups for the benchmark problems are presented in Figure 10 and Figure 11, respectively. It can be seen that the parallelizing results in a reduction in planning times, and that the speedups are linear, and sometimes even superlinear. Superlinear speedup is possible because the sequential and the parallel version use one global and multiple local clocks, respectively. This causes unavoidable differences between the sequential and asynchronous parallel algorithm. Three out of four planning times were below 5 seconds. Only the benchmark problem DETOUR needs about 20 seconds [19].

5 Conclusion and future work

In this paper, we have introduced a new approach to parallel path planning for industrial robot arms with 6 DOF. The algorithm works in an implicit and discretisized C-space and collision are detected in the Cartesian workspace using distance computation. This avoids the time and memory consuming obstacle transformation and C-space calculation. The method is based on the A*-search algorithm and needs no essential off-line computation. This approach enables the path planner to work reasonably fast for on-line provided environments.

The parallelization with static load balancing results in a balanced load distribution and shows very good speedups. Further acceleration of the path planner is possible by distributing communication, which can be done using a mesh-based communication network [18]. The planning strategy can also be enhanced by a multi-directional search [8].

Based on these results, we focus next on extending this path planner to be able to cope with moving obstacles, such as other robots. With some modification, our approach is also suitable for tasks in the area of virtual engineering. Instead of planning the path for robots, we are able to search a path for the sub-components which have to be mapped onto another object.

To reduce the enormous size of the search space, we are currently working on hierarchical on-line discretization of the C-space [12]. Additionally, we are developing a path smoothing method for executing the computed trajectories on real robots [1].

Acknowledgment

This work is part of the project "Scalable algorithms for parallel motion planning in dynamic environments" funded by the German Basic Research Framework (DFG-Schwerpunktprogramm) "Efficient algorithms for discrete problems and their applications".

References

[1] Bordon U.: "Parallele Glättung von Robotertrajektorien", Bachelour's Thesis, Insitute for Real-Time Computer Control Systems and Robotics, University of Karlsruhe, 1998.

[2] Fiorini P., Shiller Z.: "Time optimal trajectory planning in dynamic environments", IEEE Int. Conf. on Robotics and Automation, vol. 2, pp. 1553-1558, 1996.

[3] Fujimura K.: "Motion planning in dynamic environments", Berlin, Heidelberg, Springer, 1991.

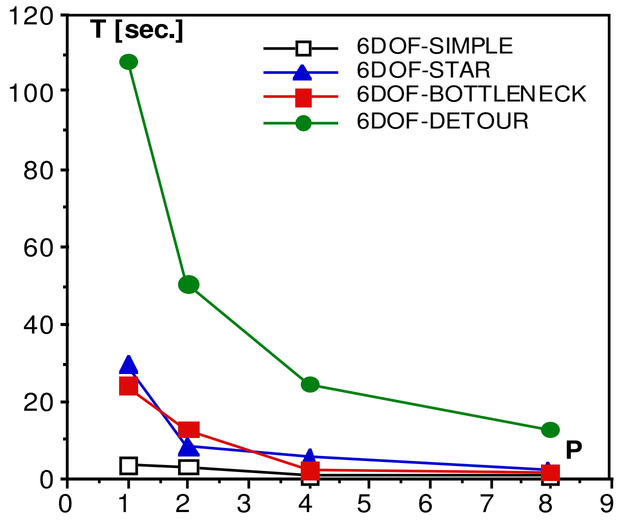

Figure 10: Planning time *T* for *P* processors solving different benchmark problems

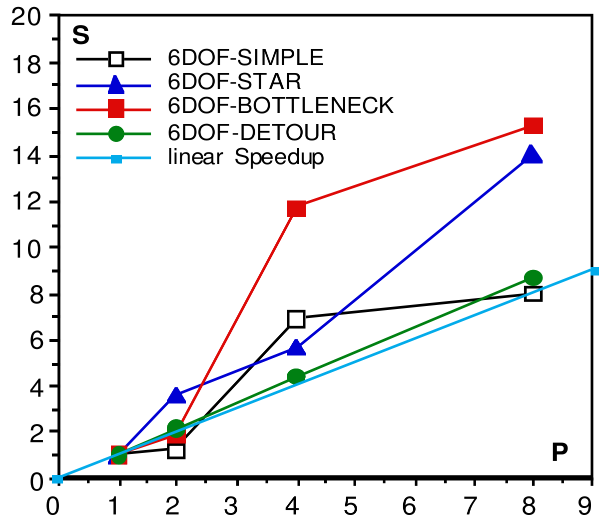

Figure 11: Speedup *S* for *P* processors solving different benchmark problems

[4] Hart P.E., Nilsson N.J., Raphael B.: "A formal basis for the heuristic determination of minmum cost paths", IEEE Trans. Syst. Sci. Cybern, pp. 100-107, 1968.

[5] Henrich D., Cheng X., "Fast distance computation for on-line collision detection with multi-arm robots", IEEE Int. Conf. on Robotics and Automation, Nice, France, May 10-15, pp. 2514-2519, 1992.

[6] Henrich D., Gontermann S., Wörn H.: "Kollisionserkennung durch parallele Abstandsberechnung". In: 13. Fachgespräch Autonome Mobile Systeme (AMS'97), Stuttgart, October 6-7, 1997, Springer-Verlag, Reihe "Informatik Aktuell".

[7] Henrich D.: "Fast motion planning by parallel processing – A review", In: Jour. of Intelligent and Robotic Systems, vol 20, no 1, pp 45-69, September 1997.

[8] Henrich D., Wurll Ch., Wörn H.: " Multi-directional search with goal switching for robot path planning ". In: The 11th Int. Conf. on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems, Castellan, Spain, July 1-4, 1998.

[9] Hwang Y. K., Ahuja N., "Gross motion planning – A survey", ACM Computing Surveys, vol 24, no 3, September 1992.

[10] Kamal L., Gupta K., del Pobil, A.P.: "Practical motion planning in robotics: Current approaches and future directions", IEEE Robotics & Automation Magazine, December 1996.

[11] Katz G.: "Konzeption einer Entwicklungsumgebung unter ROBCAD für die parallele Bewegungsplanung", Master's Thesis, Computer Science Department, University of Karlsruhe, 1996.

[12] Osterroht T.: „Hierarchische parallele Bewegungsplanung für dynamische Umgebungen“, Master's Thesis, Computer Science Department, University of Karlsruhe, 1998.

[13] Qin C., Henrich D.: "Path planning for industrial robot arms - A parallel randomized approach", In Proc. of the Int. Symp. on Intelligent Robotic Systems (SIRS´96), Lissabon, Portugal, pp. 65-72, July 22-26, 1996.

[14] Ralli E., Hirzinger G.: "A global and resolution complete path planner for up to 6 DOF robot manipulators", IEEE Int. Conf. on Robotics and Automation, Minnesota (ICRA'96), 1996.

[15] Sandmann St.: "Entwicklung eines parallelen Bewegungsplaners", Master's Thesis, Computer Science Department, University of Karlsruhe, 1997.

[16] Verwer B. J. H.: "A muliresolution work space, multiresolution configuration space approach to solve the path planning problem", IEEE Int. Conf. on Robotics and Automation (ICRA'90), 1990.

[17] Wörn H., Wurll Ch., Henrich D.: "Automatic offline programming and motion planning for industrial robots". In: The 29th Int. Symp. on Robotics, Birmingham, United Kingdom, April 27-30, 1998.

[18] Wurll C., Henrich D.: "Ein Workstation-Cluster für paralleles Rechnen in Robotik-Anwendungen". In: Proceedings der 4. ITG/GI-Fachtagung Arbeitsplatz-Rechensysteme (APS'97), Universität Koblenz-Landau, May 21-22, 1997, pp 187-196.

[19] Wurll Ch., Henrich D., Wörn H.: "Parallele Bewegungsplanung in dynamischen Umgebungen", Technical Report 20/97, Computer Science Department, University of Karlsruhe, 1997.