The Z^3–Method for Fast Path Planning in Dynamic Environments

Boris Baginski
Real–Time Systems and Robotics Group, Fakultät für Informatik
Technische Universität München, Orleansstr. 34, 81667 Munich, Germany
e-mail: baginski@informatik.tu-muenchen.de

Abstract

We present a method to plan collision free paths for robots with any number of degrees of freedom in dynamic environments. The method proved to be very efficient as it ommits a complete representation of the high dimensional search space. Its complexity is linear in the number of degrees of freedom. A preprocessing of the geometry data of the robot or the environment is not required. With the time as an additional dimension of the search space it is possible to use the method in known dynamic environments or for multiple robots sharing a common work space.

Keywords: Path Planning, Dynamic Environments, Randomized Algorithms

1 Introduction

The method presented in this paper is part of our research on intelligent and autonomous robot systems. An important component of such a system is a path planner that creates collision free and physically possible motions between positions that were planned on a higher, more abstract level.

A path planner for practical use must be able to work fast in any kind of environment and with any load of the robot. For example, when operating a manipulator on a mobile platform in a known environment, the geometry of the manipulator's workspace gets known when the platform stops and the precise position is sensed. Now the manipulator should be able to work almost immediately. In addition, there are dynamic changes in the environment that makes it necessary to take time into consideration for the motion planning, e.g. when using the mobile manipulator while the platform is in motion.

The Z^3–method (ZZZ is a german abbreviation with the meaning of *goal–directed and randomized planning in temporally changing environments*) avoids preprocessing and reduces complexity as far as possible to find a path with high efficency. The planned path is not optimal, but more important is the speed and the general applicability of the scheme.

2 The Z^3–Method

The only requirement for the use of the Z^3–method is a geometric and kinematic model of the robot and the environment. In industrial applications this data is available. For service robots, sensors, for example range sensors, can be used to scan the geometry of the environment.

The method is a further development of the ZZ–method by B. Glavina [6]. It consists of two hierarchically coupled components. The lower level is an efficient goal–directed planner that only uses local information to try to pass obstacles. The upper level is a randomized planner that uses the local planner and combines the results. The global planer explores the whole search space heuristically.

Local Goal–Directed Planning

The pose of a robot can be described by the positions of its joints if the geometry is known. One point in the space of possible joint values (the so called *configuration space*, short *c-space*) is the precise description of a robot's position. The solution of the path planning task for an n-joint robot is to find a curve between start and goal in the n-dimensional *c-space*. In a dynamic environment the *c-space* is extended by the dimension time and the solution has to be found in the $n+1$-dimensional *c-space-time*. This extension is not hogenous, as the dimension time is bound to strictly monotonous increase, while the other dimensions allow any motions, only constrained by the robot's dynamics.

Local Planning in Static Environments

To avoid exponential complexity with respect to the number of dimensions no complete representation of the search space is constructed. In contrast, only one dimensional subspaces are explored. The goal directed search moves linear from start to goal. The motion is calculated in discrete steps, collision checks are performed in short distances. The stepwidth is calculated from the tolerance that was added to the environment (or robot) geometry model and assures collision free continuous motion between two test points [6].

If a collision with an obstacle occures an avoiding movement (*slide step*) is tried. First a point that is very close to the obstacle's surface is calculated with a depth-limited bisection. Then directions that are orthogonal to the desired direction and orthogonal to each other are computed. These directions and the respective reverse directions are the possible avoiding directions. In the n-dimensional case this results in $2(n-1)$ possible directions. Figure 1 (a) illustrates this calculation. The underlying assumption is the local flatness of the $(n-1)$-dimensional surface of the *c-space* obstacles.

The stepwidth for the slide step is now calculated to not exceed the discretizing stewpwidth mentioned above. Then the avoiding points that leads closer to the goal are checked for collision. If none of the allowed avoiding points is collision free the local planner terminates without success (dead end). The restriction on steps that leads closer to the goal is necessary to avoid 'fluttering' that leads into an infinite loop. This method can be interpreted as the motion in a potential field with its global minimum at the goal. Only decreasing motions are allowed. Dead ends can be seen as local minima.

After a succesful slide step the linear motion in the direction of the goal is tried again, and again it may be necessary to calculate a slide step. Figure 1 (b) shows a sequence of slide steps along an obstacle's surface.

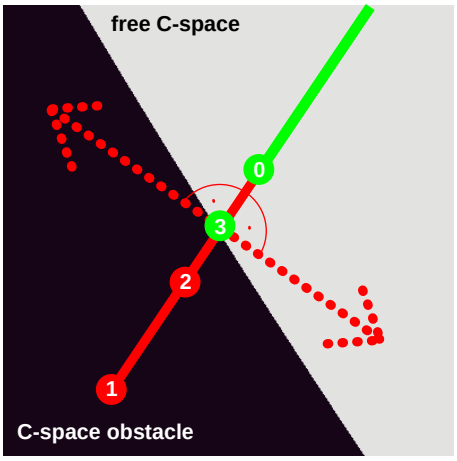
In many cases the proposed combination of goal directed steps and slide steps can find a solution for the path planning task by only considering local information. Figure 1 (c) shows an example. In static environments time is irrelevant and the local planner can retry a failed task in the reverse direction from goal to start. In many cases dead ends can be avoided with reverse search.

Local Consideration of Time

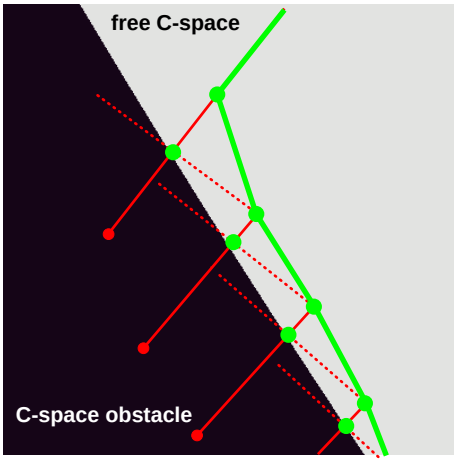
Time can be included as an additional dimension in the local planning process if the following requirements are fulfilled:

- the starting time is known
- the status of the system can be calculated at any later time (the dynamics of the environment are known)
- for all discretizing steps, the time of the current step can be calculated from the time of the last step or a limited number of earlier steps (the dynamic behaviour of the robot is known).

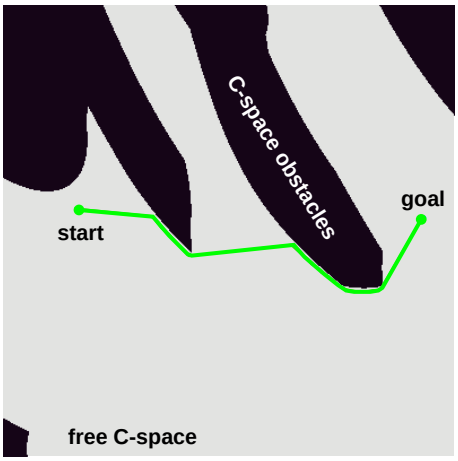
Starting the local planner from a known point in *c-space-time* allows to plan a path and an arrival time. For the application in reality the transition from linear steps to slide steps — resulting in a sudden change of direction — must be considered with care. This can be achieved by correcting the time of some earlier steps to slow down the motion appropriately. The positions moved in *c-space-time* then have to be checked again for collision, this may result in an earlier start of the slide steps. In the general case, the adjustments for the directional change are limited to a fixed number of steps and happen only locally.



(a)



(b)



(c)

Figure 1: (a) *The last collision free position of the linear movement is* **0**, *the next step would be* **1** *but collides. With a bisection (points* **2** *and* **3***) a point very close to the surface is calculated and orthogonal avoiding directions are computed.* (b) *A sequence of successful slide steps along a c-space obstacle.* (c) *An example for a path planned with the local planner as a combination of goal directed and slide steps.*

Global Randomized Planning

First of all, it's tried to solve the task with the local planner. If this fails, random collision free subgoals are used to create partial tasks that can solve the whole task through combination. There are two parameters to be chosen for the global planner, first the maximum number M of subgoals that are used to find solutions, and second the maximum number m of subgoals that are allowed on a path from start to goal.

When using the ZZ-method [6] it shows that most of all tasks can be solved with one or very few subgoals along the path. It proved to be more efficient to restart the planner with M new random subgoals instead of allowing any number ($< M$) of subgoals on the path. For the same reason it is better to check the possible paths with one subgoal first, then the paths with two subgoals and so on. As far as possible unnecessary local planning shall be avoided. The following algorithm follows this conception:

```
check the direct path from start to goal, if
its possible then SUCCESS
create M random subgoals as the initial
members of set U (set of unconnected points)
initialize the set S (points that can be
reached from the start) with the starting
point.
Loop from 1 to m
    initialize the set S_new as empty set
    Loop for all points s_i out of S
        Loop for all points u_j out of U
            try to connect s_i with u_j with the
            local planner
            if this is successful:
                try to connect u_j with the goal
                with the local planner
                if this is possible:
                SUCCESS
                if this is not possible:
                    remove u_j from U
                    add u_j to S_new
        end Loop
        remove s_i from S
    end Loop
    make S equal with S_new
    if S is empty, then NO SOLUTION
    RETRY WITH NEW SUBGOALS
end Loop
NO SOLUTION
RETRY WITH NEW SUBGOALS
```

This algorithm creates a tree from the start in the set of subgoals. For every height of the tree possible connections to the goal are tested before the growth continues. This principle is illustrated in figure 2. In static environments the global planning can start simultanously from the goal. This results in a faster reduction of the number of unconnected points and thus in faster planning. An example for the whole planning process is shown in figure 3.

Global Consideration of Time

The time can be included in the global planning as well, if the starting time and the dynamics are known. In this case the subgoals are not fixed in the time when they are created. The time is fixed when the subgoal is reached with the local planner, where the time is calculated incrementally in discrete steps. Not being fixed in time, the subgoals are not guaranteed to be collision free, but the random generation can estimate the time heuristically and thus reduce the danger of colliding subgoals. The time is propagated forward in the growing tree, every subgoal is reached only once with the local planner yielding a subgoal arrival time. In the end, a goal arrival time can be returned. It is not possible to plan from goal to start in dynamic environments.

Complexity and Completeness

The presented algorithm evaluates all paths with up to m subgoals. This testing is complete, so if such a path exists for the given random subgoal distribution it is found. The restriction of the search depth to a constant value results in a complexity that is linear with respect to the number of M, the subgoals used in total. The complexity of the local planner is linear with respect to the number of degrees of freedom. This number determines the number of avoiding directions that have to be tested.

The underlying assumption of the Z^3-method is that there exists a large number of possible paths and one of those can be found with random subgoals with high probability. The search space is examined in very small parts only. In complex environments, only a large number M of possible subgoals may lead to a success. The maximum amount of time to find an existing solution is unbound. If M and m would grow unlimited, the method would be probabilisticly complete: if there exists any solution, it is found.

For the practical use these properties or the Z^3-method are of no meaning. It shows to be very efficient in its conception, because in most cases a path can be planned in short time. The durations for planning in realistic environments are fractions of seconds or a few seconds. Only very complex tasks that can not be solved with complete planners due to complexity may take minutes or hours.

3 Practical Results

All experiments shown in this chapter were performed on a Hewlett Packard Unix Workstation 9000/730. Only the user time of the planning process was measured. The geometry simulation was developed in our laboratory. It

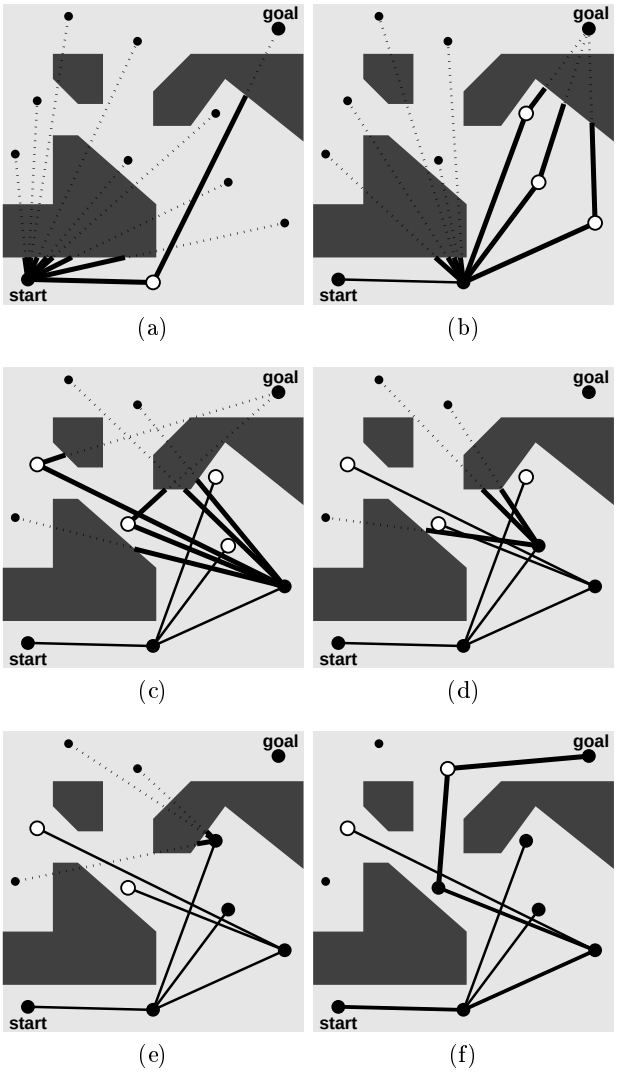


Figure 2: *Systematic illustration of the global planning (there are no slide steps shown for clarity). The direct connection between start and goal fails, so $M = 9$ random subgoals are created. In picture* (a) *all possible connections with one subgoal are checked. In just one case a connection between the start and an unconnected point is found. It is tested for a direct connection to the goal immediately. This fails, the point itself is the first leaf of the tree growing from the start. Beginning at this point, all connections with two subgoals are evaluated in* (b)*, resulting in three new leaves. They are expanded in* (c)*,* (d) *and* (e)*, thus testing all possible connections with three subgoals. In* (c) *two new leaves are created,* (d) *and* (e) *result in dead ends. The decreasing computational costs for the tests, resulting from the reduced number of unconnected points, can be seen well in this example. In* (f) *a solution for the path planning problem is found with four subgoals. Given this random distribution, this is the connection with the smallest number of subgoals, found with the smallest number of local planning.*

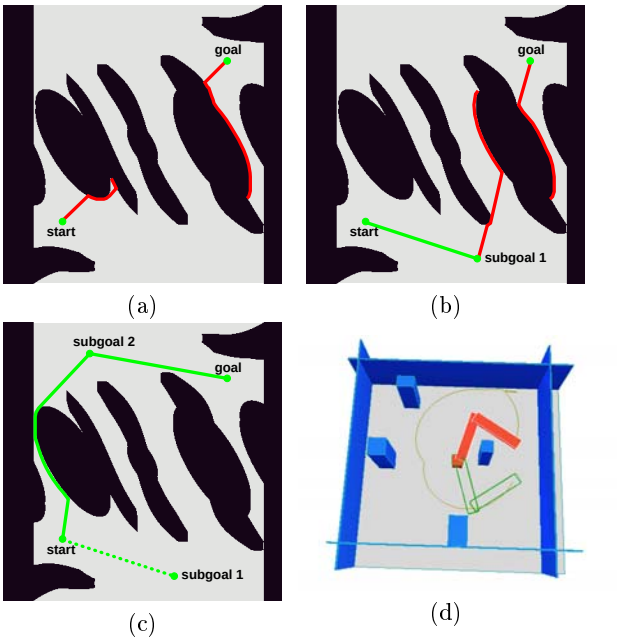


Figure 3: *An example for the planning process in a static two dimensional environment shown in* (d)*. In the* c-space *map in* (a) *the failure of the local planner starting from start and from goal can be seen. In* (b) *a first subgoal is tested that can be reached from the start but fails to be connected with the goal. In* (c) *the second subgoal tried leads to a solution. The planned path is shown in* (d) *as the movement of the tool center point.*

is based on an automatically created hierarchy of hull bodies that allows very efficient collision testing. Up to now, our system is only capable to plan paths for robots in static environments.

Scenario MOBROB

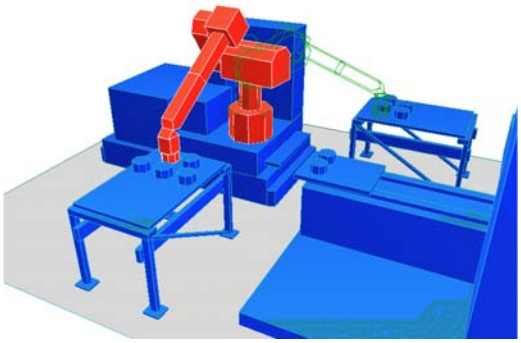


Figure 4: *MOBROB scenario*

This scenario shows a manipulator on a mobile platform in an industrial environment. For the experiments only the six joints of the manipulator are used (the platform is static). The six dimensional *c-space* contains 57.6% free space. The simulation system needs an average time of 0.4 ms to check a position for collision. The experiments are performed with $M = 25$ total sub-

goals and a maximum of $m = 4$ subgoals on one path. 5,000 tasks were created by combining random positions that are very close to obstacle surfaces, so these tasks can be described as random pick-and-place–tasks. The following results are measured:

- 100% of the tasks are solved
- the average run time is 0.064 sec, max. is 2.56 sec
- there is an average of 0.042 subgoals per planned path. This shows the efficency of the local planner, that can solve almost all tasks
- the local planner is used 1.16 times on an average.

Especially the results in this very realistically modelled scenario prove that the Z^3-method is a method of choice for path planning in real applications with hard constraints on time. Even in the rare cases were the global planner becomes necessary the planning times are usually below one second.

Scenario ROTEX

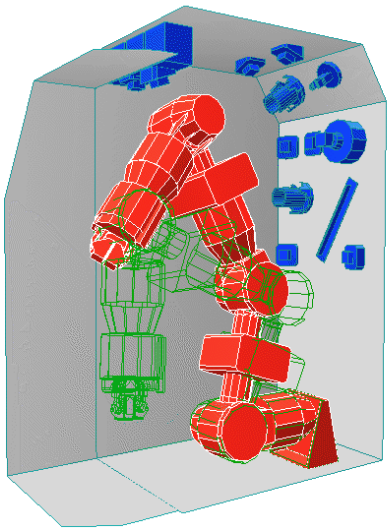


Figure 5: *ROTEX scenario (front walls removed)*

The ROTEX–workcell was used for a number of robotic experiments in space on board the space shuttle mission D–2 [7]. The real geometry data from the DLR is used for the simulation. The workcell is very small, the six dimensional *c-space* contains only 1.35% free space. The simulation system needs an average time of 0.8 ms to check a position for collision. Both experiments were performed with $M = 25$ total subgoals and a maximum of $m = 4$ subgoals on one path.

In the first experiment the task shown in figure 5 is solved 100 times. The difficulty of the task is not obvious in the picture. The lower half of the arm has to be turned completely. The following results are measured:

- the average run time is 42.65 sec, max. is 117.55 sec
- only an average of 17.65 subgoals out of the 25 are 'touched' by the local planner (because a solution with one subgoal is found)
- there is an average of 1.49 subgoals per planned path
- the local planner is used 26.87 times on an average
- the ratio between the length of the planned path and the length of the path explored with the local planner is 8.13.

In the second experiment 5,000 random tasks (combinations of two random positions) are planned. The following results are obtained:

- 99.54% of the tasks are solved
- the average run time is 3.68 sec, max. is 114.76 sec
- there is an average of 0.35 subgoals per planned path. This shows the efficency of the local planner, that can solve about two third of all tasks even in such a small workcell
- the local planner is used 4.45 times on an average.

Scenario GRIPPER

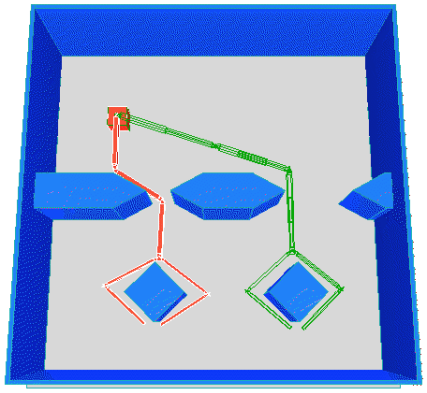


Figure 6: *GRIPPER scenario*

The scenario in figure 6 follows a scenario used in [8]. The robot has 10 degrees of freedom. These are three translatoric joints and three rotational joints in the arm and two rotational joints in the two 'fingers'. The left slot is very narrow and to move the arm from the left position into the upper free space is all but trivial. The *c-space* contains only 3.8% free space. A collision check takes an average of 0.47 ms in the simulation. The planner is started with $M = 500$ total subgoals and a maximum of $m = 4$ subgoals on one path. In the experiment the task shown in figure 6 has to be solved 30 times. The following results are measured:

- the average planning time is 23,220 sec (ca. 6.5 h), max. time is 98,400 sec (ca. 27 h)
- on an average, the global planner restarts 2 times per run and creates 500 new random subgoals
- an average of 1,316.7 subgoals are 'touched' by the local planner
- there is an average of 3.13 subgoals per planned path
- the local planner is used 12,741 times on an average
- the ratio between the length of the planned path and the length of the path explored with the local planner is 3,911.

The results achieved in this scenario show that the Z^3-method is not only theoretically but as well practi-

cally 'probabilistic complete'. Even for very complicated tasks with a very high dimensional search space a path can be planned.

4 Future Work

The global planner repeatedly uses the local planner, but in many cases these local plannings are independent from each other. Based on this observation a concept to parallelize the planner on a network of workstations was develeoped and is currently implemented.

The run time of the Z^3-method is coupled with the efficency of the geometric simulation. In this area we are developing new efficient data structures and algorithms and try to use results from the growing research area 'computational geometry'.

One problem of the local planner is that the slide steps are very close to the obstacles, yielding 'dangerous paths' in uncertain environments. We try to integrate dynamic *protection shields* to guarantee a safety distance whenever this is possible. This has to be done without decreasing the planners efficency and without increasing its conceptual complexity.

Further investigations are done in developing a new local planer that is especially efficient for hyperredundant manipulators. By *shrinking* and *expanding* the model of the robot along its trajectory its possible to find solutions in much more cases than with the slidesteps, with the same linear computational complexity. First results are very promising [1].

A modification of the Z^3-method is used for path planning for cooperating manipulators with dependent tool coordinate systems, e.g. cooperative transportation [5]. These conceptions shall be extended and generalized to plan all classes of tasks for cooperating manipulators with overlapping workspaces.

5 Discussion and Conclusion

We presented a method that is able to plan paths in dynamic environments. Its complexity is linear in the number of degrees of freedom. The usability of the method was demonstrated in several scenarios. Some problems and objections shall be discussed in this chapter.

The path planned with the Z^3-method is not optimal. The use of subgoals results in sharp corners and detours in the path. For static environments we use a local polygon optimizer that improves the quality of the pathes very efficiently [3].

In static environments the Z^3-method can not use the 'experiences' of prior planning. The 'subgoal–tree' is removed when a solution is found. Other path planning algorithms that represent the free part of the *c-space* with a graph, e.g. [4, 8], or with cells, show better results for repeated planning in the same environment. In our opinion, the dominating criterium is the constant efficency in unknown and dynamic environments that is a property of the Z^3-method. The only planner we know with comparable good results is the *R*andommized *P*ath *P*lanner [2, 9, 10], but its goal-directed component requires an expensive precomputation of the workspace to calculate a potential field, and the so called 'random walks' that escape local minima will not use the whole free space in a way the random subgoals do.

To summarize, the Z^3-method is a reliable and versatile concept. The global planner can be used in other areas as well and is not bound with robotics. All planning tasks that can not be solved in one step and where the decomposition is not obvious can be adressed with this strategy.

References

[1] Boris Baginski. Local motion planning for manipulators based on shrinking and growing geometry models. In *Proceedings of IEEE Conference on Robotics and Automation (submitted)*, Minneapolis, April 1996.

[2] J. Barraquand and J.-C Latombe. A monte-carlo algorithm for path planning with many degrees of freedom. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 1712–1717, Cincinnati, Ohio, May 1990.

[3] Stefan Berchtold and Bernhard Glavina. A scalable optimizer for automatically generated manipulator motions. In *Proccedings of IEEE/RSJ/GI International Conference on Intelligent Robots and Systems IROS'94*, pages 1796–1802, Munich, September 1994.

[4] Martin Eldracher. Neural subgoal generation with subgoal graph: An approach. In *Proceedings of World Conference on Neural Networks WCNN '94*, 1994.

[5] Max Fischer. Efficient path planning for cooperating manipulators. In *Proccedings of IEEE/RSJ/GI International Conference on Intelligent Robots and Systems IROS'94*, pages 673–678, Munich, September 1994.

[6] Bernhard Glavina. *Planung kollisionsfreier Bewegungen für Manipulatoren durch Kombination von zielgerichteter Suche und zufallsgesteuerter Zwischenzielerzeugung.* PhD thesis, Technische Universität München, February 1991.

[7] G. Hirzinger. Rotex – the first space robot technology experiment. In *Preprints of the Third International Symposium on Experimental Robotics*, pages 302–320, Kyoto, Japan, October 1993.

[8] Lydia Kavraki and Jean-Claude Latombe. Randomized preprocessing of configuration space for fast motion planning. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 2138–2145, San Diego, California, May 1994.

[9] Jean-Claude Latombe. *Robot Motion Planning.* Kluver Academic Publishers, 1991.

[10] Jean-Claude Latombe. Geometry and search in motion planning. *Annals of Mathemathics and Artificial Intelligence*, 8:215–227, 1993.