EFFICIENT MOTION PLANNING IN HIGH DIMENSIONAL SPACES: THE PARALLELIZED Z^3-METHOD

Boris Baginski

Technische Universität München — Robotics and Real-Time Systems Group
Orleansstr. 43, D–81667 München, Germany
e-mail: baginski@informatik.tu-muenchen.de

ABSTRACT – We present a method to plan collision free paths for manipulators with any number of degrees of freedom. The method is 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. In the planning process, several more or less independent partial tasks of differrent complexity can be identified, thus allowing to parallelize the algorithm in several ways to increase the efficiency towards real-time operation in most practical cases. This paper gives an overview of our recent concepts and implementations.

KEYWORDS: Robotics, Path Planning, Parallel Algorithms, Autonomous Robots

1 Introduction

The Z^3-method is part of the research efforts of our group to develop intelligent and autonomous robot systems that can be commanded on the task level. An important part of such a system is a motion planner that connects positions that are required by a higher level planning system with collision free and physically possible trajectories. Within the presented project we want to use parallel computation to increase the efficiency of the planning[1].

Robot motion planning is a very complex problem. Even if we consider the case of one moving robot in a static environment only, the dimensionality of the search space for the robot's motion equals the number of its kinematic degrees of freedom (DOF). Any universal robot needs at least six joints to operate unconstrainted in its workspace. The space of possible joint values of a robot is called its configuration space (c-space). All positions that result in collisions with the workspace obstacles imply the c-space obstacles. All possible robot motions are curves in free c-space [7].

There are several different approaches to motion planning. All attempts to build up complete maps of the c-space fail for more than 4-5 degrees fo freedom [8]. Even at a coarse resolution, the exponential complexity (in time and space) prohibits this kind of approaches for real robots. Another concept is to build up a randomized graph in the c-space [6, 4]. A given number of subgoals are placed randomly and connected in a neighbourhood by simple local search strategies (e.g. check linear connection for collision). Thus the topology of the c-space can hopefully be covered without exponential complexity. After this preprocessing, motion planning means just finding connections from start and goal to the graph, and graph search. But the required preprocessing may take very long for complex environments (at least in the order of minutes or hours).

We are seeking motion planning algorithms that can take and immediately use the geometry data, as this data often gets available when instant planning required. This is the case for service robots that use sensors to build up an environment model, and another example are mobile manipulators in industrial environments. All geometry is known, but as the position of the manipulator arm itself is time dependent, no c-space map can be constructed. Another requirement for our motion planning system is the ability to handle a high number

[1]This work is supported by the Deutsche Forschungsgemeinschaft, number SI 364/2-1.

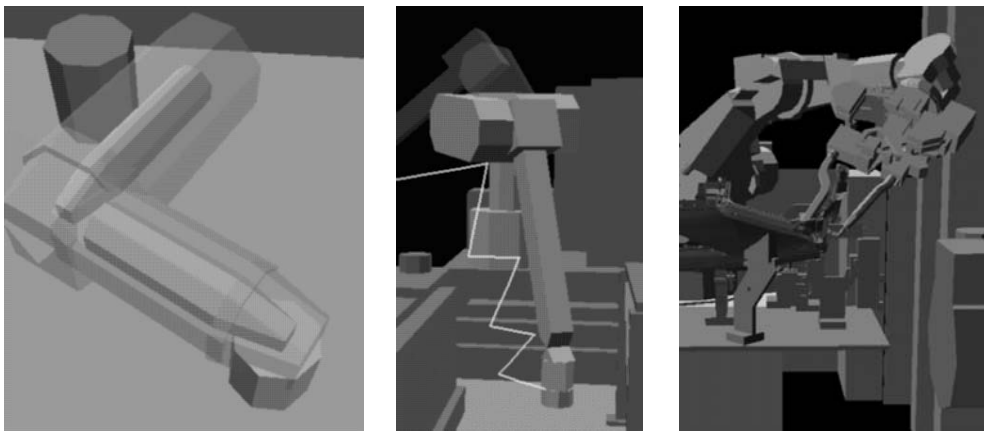

Figure 1: A: The geometry model is expanded to reduce the number of collision tests. B: An example of a path planned with slide steps in a 6 DOF environment. C: A very complex environment and robot modell in an industrial environment.

of degrees of freedom, e.g. to plan motions for the manipulator and its mobile platform simultaneously.

Another possibility to address the motion planning problem are the local approaches. The idea is to move from the start position towards the goal, and to pass obstacles along the way, only considering 'local' information. One method is the so-called potential field approach [7]. The goal implies an attractive force, the obstacles are repulsive. The robot moves in this force field, following the gradient. The major drawback of this method is to get stuck in local minima of the potential field, thus requiring random escape or random exploration techniques to avoid the complexity of global search [2].

2 The Z^3-Method

The Z^3-Method uses a hierarchical combination of a modified potential field planner and random global exploration. The lower level local planner tries to *slide* along obstacles to pass them, the upper level uses randomly placed subgoals to overcome local minima of the local planner, avoiding the high complexity of global search. The method was developed a few years ago and is continously improved [5, 1]. In the following, we give an overview of our current path planning system. In the third section, we discribe our algorithm to gain maximum efficiency with parallelization.

The only requirement for the Z^3-method is a geometric and kinematic model of the robot and its environment. This is used to check single positions and segments of paths for collision, as we only implicitely plan in the c-space. The efficiency of these tests are the key factor for the performance of the overall planning system. Collision detection is the lowest level and described in the next paragraph, thereafter we explain the local planner and the global planner. The quality of the planned paths is increased through local optimization, the principles are explained in the fourth paragraph of this section.

2.1 Collision Detection

A manipulator is composed out of a chain of rigid parts, connected with joints. To check parts for collision with the environment, we use a hierarchical representation of the environment. All facets of the geometry model are enclosed by an axis-parallel bounding box (that is represented with just two points). These boxed facets are the leaves of a binary tree of hull boxes that is constructed bottom-up. At any step of the construction, the two nodes that have the smallest increase of volume when linked together are united to form a new node of the tree. Collision detection is executed by intersecting the bounding box of the moving part with the tree, beginning at the root node. All child trees of intersecting nodes are tested recursively. This test is very fast if the parts are at a certain distance, and it is fast as well to locate the possibly intersecting facets at the bottom of the tree.

For path planning, it is not sufficient to check single positions for collision, but whole paths. In principle, this would require an infinite number

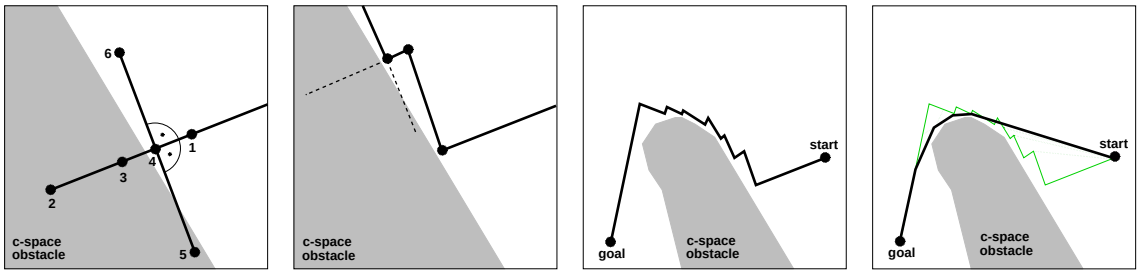

Figure 2: A: Local search for collision-avoiding positions to execute a slide step. B and C: Repeated slide steps can solve the planning locally. C: The path is smoothed through local optimization.

of tests along the path. Two possibilities exist to avoid this. One is to use distance computation at each position, and to ensure that no point of the concerned robot part moves further than the measured distance in the next step. The major drawback is the expense of this kind of test, as it is much more complex than simple collision detection. The other possibility is to enlarge the geometry model of the robot to ensure a *protective shield* of a certain thickness d. If the robot part does not collide at two positions that are less or equal $2d$ apart, the straight motion inbetween these two positions is collision free.

We use the latter principle. The motion of the robot is not discretized and all parts are checked at intermediate positions, but the motion of each part of the robot is discretized individually. In general, the links close to the base move shorter distances than the links closer to the tool. We use an approximation scheme that calculates a joint step with an upper bound for the cartesian motion of a link out of the cartesian motion and the joint step in the previous discretization step. This ensures collision free paths with a minimum number of collision tests.

To further reduce the number of collision tests, we use several expanded models with different *thicknesses* for the protective shield. Up to now, the shield construction is applicable only for simple geometry models, but these tests prove the potential of this concept, as it is able to reduce the number of collision detections to 10%, compared to fixed size shields. Fig. 1 A visualizes high level shields for a 2 DOF robot.

2.2 Local Planning

The local planner tries to move the robot on a straight line in c-space from start towards goal. If a collision occurs, the exact point of intersection is approximated through a depth-limited bisection. Then the planner calculates and tests avoiding steps. The directions are chosen orthogonal to the current straight direction and orthogonal to each other. Thus they are the positive and negative base vectors of a hyperplane, constructed orthogonal to the straight motion in the collision point. For a robot with n degrees of freedom, their number is $2(n-1)$, resulting in a linear complexity. The step length of the avoiding steps is chosen heuristically and proportional to the thickness of the protective shield. Fig. 2 A illustrates this computation in a 2-dimensional c-space. The last discretized step is 1, 2 would be the next but collides. Bisection (3 and 4) finds a point close to the surface. Orthogonal directions are calculated and tested (5 and 6).

If a collision free avoiding position is found and can be reached, the planner starts again to move straight towards the goal. If neccessary, additional *slide*-steps are executed (see fig. 2 B and C). The local planning reaches the goal in many cases, but it may get stuck if it can not find an avoiding position that is closer to the goal than the last straight line starting point or none of the avoiding positions is collision free (dead end). If there are multiple protective shields and the current active shield is not yet the base layer, the level is reduced and planning is continued. If everything fails, the reverse direction from goal to start is planned, as it may avoid the local minimum.

2.3 Global Planning

If local planning can not solve a task, the global planner is used to create subtasks to find a solution through combination (see fig. 3). A set of M unconnected random subgoals is created at arbitrary positions in free c-space. The local planner is called to find connections to these subgoals, one after another. Whenever a con-

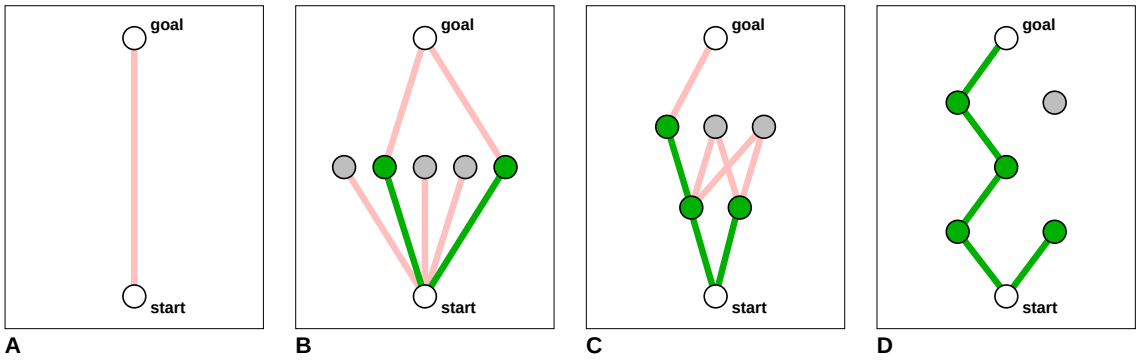

Figure 3: The global planner constructs a subgoal tree, growing from start into the set of random subgoals. Bright lines indicate failure, dark lines success of local planning.

nection to one of the subgoals is found, the connection to the goal is tested. In most cases, a solution can be found with one subgoal. But if this fails, all subgoals that were reached from the start are tested for connections with all unconnected subgoals, and again, if a connection is found, the connection to the goal is planned locally. If necessary, this procedure can be repeated, and it can be interpreted as a tree that is growing into the subgoal set. The search depth is limited, and if it is reached or no further leaf can be found, the global planning terminates with failure.

In realistic environments, all tasks could be solved with M=25 random subgoals and a search depth limited to 4 [1]. The planning time is mostly dependent on the complexity of the geometry model, and, of course, the task difficulty. The planning is not complete (probabilistic completeness would be possible with an unlimited number of subgoals and an unlimited search depth), but it is sucessful in all practical cases it was applied to. It is not suitable to solve maze-like problems, but in typical manipulator environments with large areas of free space, solutions can be found reliably and fast. The computational complexity of the global planner is linear in the number of subgoals, and as this is fixed, the overall complexity of motion planning with the Z^3-Method is linear in the number of degrees of freedom.

2.4 Optimization

A planned path is not optimal, because the slide steps create a jagged motion close to the obstacles, and the random subgoals lead to detours. We have developed a very efficient scheme to smooth a path based on local optimization [3]. Only very simple polygon manipulations are used. In a first attempt, the slide steps are straightened by replacing them as far as possible by longer line segments. In the following iteration, every set of three corner points of the path is examined repeatedly. If the direct connection is possible, the inner point is removed. If there is a collision, the two sides of the triangle are intersected in the center and the resulting connection is tested. This bisection is repeated until a suitible connection is found to cut the tip of the triangle. This whole process is repeated for all triangles in the path until it can not be continued due to obstacles or a sufficient smoothness is reached, see fig. 2 D.

3 Parallel Motion Planning

As we are unable to completely analyze the highdimensional c-space, all possible tasks and all possible szenarios, we can analyze the motion planning only qualitatively, not quantitatively. In general, many tasks are solved with local planning only, in short time. Most of the remaining tasks require one subgoal, out of a small set, planned in an order of magnitude longer time. Few tasks require two or more subgoals, yielding even longer planning times. Very few task cannot be solved. The other dominant factor of the planning time is the complexity of the environment and the robot, that can be quantified through the number of facets in the geometry model. In industrial scenarios with 500 facets we measured an average planning time of about $0.1sec$ and a maximum of $2.6sec$. In an environment consisting of 17,500 facets (see fig. 1 C) the average planning time is about $2min$, the maximum is $> 1h$.

A promising way to reduce the planning time is to parallelize the algorithm. We implemented our planning system as a parallel application

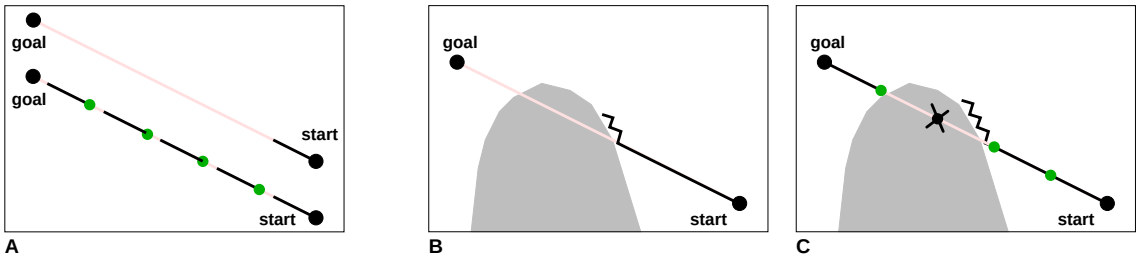

Figure 4: A: Parallel execution of straight line collision detection results in (almost) linear speedup. B and C: If there are obstacles along the straight line, some intersection points might not be accessible and the more expensive slide steps are not accelerated.

based on PVM [9], it allows us to link any number of workstations together to plan one task. Nowaday, clusters of workstations are the standard architecture, the number of computing nodes is in the order of 10^1 to 10^2. The parallelization has two main targets: maximum overall accelleration of all kinds of tasks, and best possible scalability to the number of nodes available. In the following, several parallelization possibilities are explained for local and global planning, and in the third paragraph an integrated, adaptive scheme is presented.

3.1 Parallel Local Planning

The one-way communication time in a typical ethernet environment is in the order of $10ms$, thus the smallest granularity of parallelization is at the level of at least one or a few collision tests (they run in the same order of time in complex environments and much faster in simpler environments). An obvious way to distribute the local planning process is to calculate equidistant intersection points and to plan the subtasks independently. If there is no collision along the straight line connection, almost linear speedup can be obtained (see fig. 4 A). But if there are one or more obstacles along that line, two problems occur: Some of the intersection points may be in collision, yielding *longer* partial tasks in the case where slide steps are required (see fig. 4 B,C). Thus the speedup is limited, and if a dead end occurs, several of the partial tasks closer to the goal may have been planned in vain.

Another approach is to parallelize the slide steps only. The maximum number of avoiding step tests for an n-DOF Manipulator is $2(n-1)$, the average depends on the obstacle topology, but is linear dependent on n. So we use one node for the straight motion, in case of collision all orthogonal avoiding steps are calculated and the path segments are tested by a dedicated number of nodes simultanously.

3.2 Parallel Global Planning

The global planner can be parallelized by distributing local planning tasks onto the computing nodes, scheduled by a coordinating node. The tasks are priorized, and the local planners can be interrupted to execute partial tasks of higher priority. The first step is to plan connections from the start to as many subgoals as nodes are available. If a subgoal can be reached, it is immediately tested for connection with the goal, these partial tasks have the highest priority. The next highest priority is assigned to the connections between start and each subgoal, below them are the connections between reached subgoals and unconnected subgoals. The scheduling assures that all nodes are working at all times, and that the most promising connections are checked first.

Another possibility is to run several instances of the global planner in parallel, each planning independently with its own subgoal set. The computation time for tasks that require subgoals is distributed through the random component, this kind of parallelization reduces the expected value and the variance.

3.3 Adaptive Parallelization

The local parallelization is only efficient for very simple tasks, because it requires an enormous amount of communication if several houndred or thousand runs of the local planner have to be executed to solve the task. In general, the best granularity is proportional to the task difficulty. But the task diffficulty is not known a priori. So we start with the assumption that the task is simple, and we parallelize our algorithm in a way that it can be solved as fast as possible or it can be concluded as fast as possible that

it is not simple. This is done by combining the ideas presented above: the task is intersected and assigned to groups of nodes that are used to parallelize the slide steps. If the task is not simple, we continue planning under the assumption that the task is of average difficulty, using scheduled global planning with parallelized local planning, each local planner is assigned a subset of the available nodes. If this does not solve the task, more subgoals have to be used and we run scheduled planning with complete local planning within each node. If the task is extremely difficult, we run complete global planning in each node. This scheme adapts itself to the task difficulty, thus gaining the best speedup from parallelization for each class of tasks.

4 Ongoing Work

The implementation of our system is completed, we are in the evaluation phase. Several parameters have to be adjusted for best performance, and they are as well dependent on the number of nodes available: the number of dedicated nodes to test avoiding steps, the minimum length of straight line subtasks, the number of subgoals etc. We are going to implement simple heuristics that showed good results in sample scenarios.

The scenarios we use are created from real data out of industrial production (see fig. 1 C) and robot research projects. One of this scenarios is a mobile service robot with a 7-DOF manipulator, where planning is done in the complete ten dimensional search space to enable manipulation while the platform is in motion. To evaluate the speedup, we run two classes of tests. One series is executed for a few realistic, manually created tasks. The other series uses a large number of randomly created tasks, where two positions have to be connected that are both close to collisions for the robot's tool (random *pick-and-place*–tasks).

The next steps are the parallelization of the optimizer and the development of geometric algorithms and data structures to handle several shields, thus reducing the number of collision tests for complex geometry data as well.

5 Conclusion

We presented an efficient system for motion planning in all kinds of environments, for all kinds of robots with any number of degrees of freedom. To increase its efficiency for realistically complex scenarios, we use a scalable and adaptable parallel algorithm. Currently, our implementation is completed and a large number of tests is executed to evaluate the speedup gained through parallelization. The preliminary results are very promising and show the efficiency of our approach, aiming at motion planning in real time for most of all tasks, i.e. a planning time fairly below the time the execution of the motion demands.

Bibliography

[1] Boris Baginski. The Z^3–method for fast path planning in dynamic environments. In *Proceedings of IASTED Conference Aplications of Control and Robotics*, pages 47–52, January 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*, pages II–142 – II–146, 1994.

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

[6] Lydia Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. Technical Report UU-CS-1994-32, Utrecht University, August 1994.

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

[8] E. Ralli and G. Hirzinger. A global and resolution complete path planner for up to 6DOF robot manipulators. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 3295–3302, Minneapolis, Minnesota, April 1996.

[9] Vaidy S. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency: Practice and Expirience*, 2(4), December 1990.