FAST MOTION PLANNING IN DYNAMIC ENVIRONMENTS WITH THE PARALLELIZED Z^3–METHOD

BORIS BAGINSKI
Technische Universität München, Institut für Informatik,
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. The method allows efficient parallel planning of independent sub-tasks to increase its performance.

KEYWORDS: path planning, dynamic environments, randomized algorithms, parallelized atgorithms

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. In addition, there are often dynamic changes in the environment that make it necessary to take time into consideration for the motion planning.

The pose of a robot can be described by the vector of its joint positions. One point in the space of possible joint values (*configuration space* or *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* [8]. In a dynamic environment the *c-space* is extended by the dimension time. The solution has to be found in the $n+1$-dimensional *c-space-time*. This extension is not homogenous, as the dimension time is bound to strictly monotonous increase, while the other dimensions allow any motions, only constrained by the robot's dynamics.

In this paper we present the Z^3-method for path planning as a sequential algorithm for static environments. It is then extended to dynamic environments and the way we parallelized it is introduced. The efficiency is demonstrated in two scenarios.

THE Z^3–METHOD

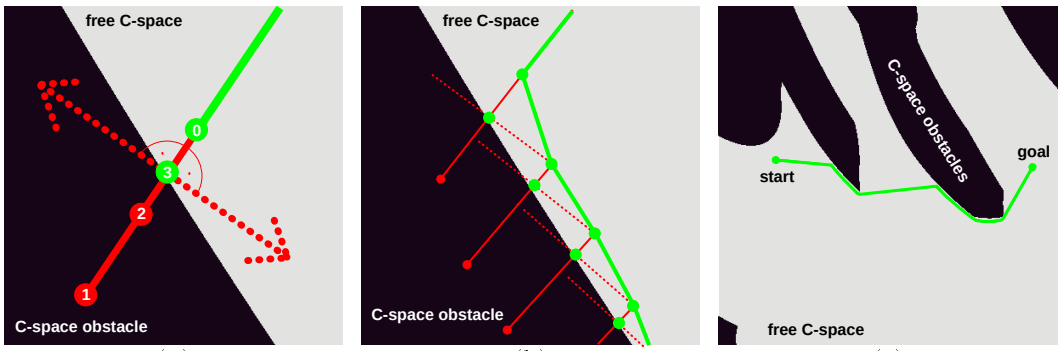
The only requirement for the Z^3–method (ZZZ is a german abbreviation of *goal–directed and randomized planning in temporally changing environments*) is a geometric

and kinematic model of the robot and 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.

Local Goal–Directed Planning

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 tests are performed in short distances. The stepwidth is calculated from the tolerance that was added to the robot's 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.



(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.*

The stepwidth for the slide step is now calculated to not exceed the discretizing stepwidth. Then the avoiding points that lead 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 lead closer to the goal is necessary to avoid 'fluttering' that results in an infinite loop.

After a successful 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 considering local information only, see Figure 1 (c). 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. In static environments time is irrelevant and the local planner can retry a failed task in the reverse direction from goal to start.

Global Randomized Planning

First of all, it is tried to solve the task with the local planner. If this fails, random collision free subgoals are used to create partial tasks to 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. Tests show that most of all path planning tasks can be solved with one or very few subgoals along the path. These tests proved that it is more efficient to restart the planner with M new random subgoals instead of allowing any number ($< M$) of subgoals on the path. 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, growing from the start into the set of subgoals. For every height level of the tree possible connections to the goal are tested before the growth continues. In static environments the global planning can start from the goal as well. This results in a faster reduction of the number of unconnected points and thus in faster planning. The presented algorithm evaluates all paths with up to m subgoals. This testing is complete, if a path exists for the given random subgoal distribution it is found.

Consideration of Time

Time can be included as an additional dimension in the local planning process if the starting time is known and the status of the system can be calculated at any later time, i.e. the dynamics of the environment and the robot are 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 (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 time can be included in the global planning as well. The subgoals are not fixed in time when they are created. The time is fixed when the subgoal is reached by the local planner. 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 reverse from goal to start in dynamic environments, neither local nor global.

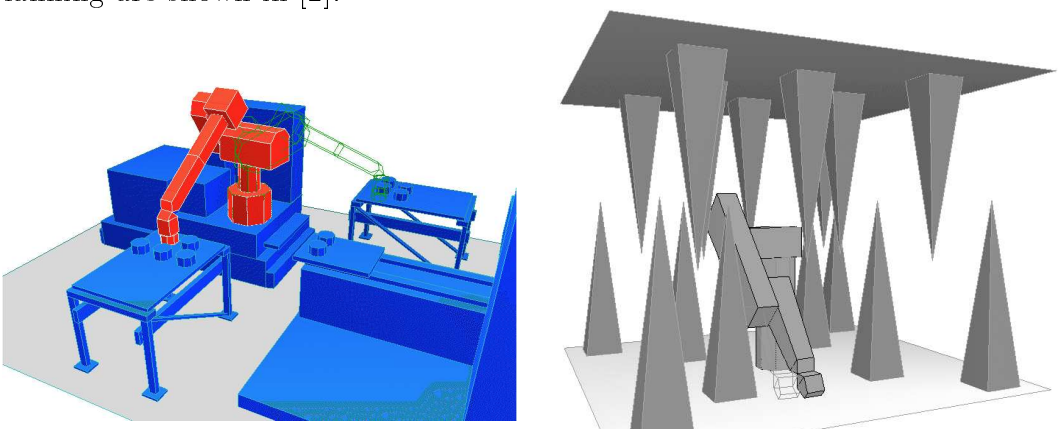
Parallelized Planning

There are of course several possibilities to parallelize the Z^3-method. The most obvious way is to run several instances of the local planner in parallel. The tasks of the local planer are almost independent of each other, only connected at start and end positions. The inner loop of the algorithm presented above is the key to efficient parallelization of the local planning. All leaves of the tree that was constructed from the start can be tested with all unconnected subgoals simultaneously.

The number of computing nodes is in general neither dynamic nor unlimited, but a fixed configuration parameter, e.g. the number of workstations available in a local cluster. Thus we propose a *scheduling* algorithm to control the available computing nodes. The idea is to assure the highest possible load and to check the most promising connections in the subgoal graph first. The scheduling algorithm fills up the computing nodes respectivly. If a computing node gets available (a previous run of the local planner has terminated), the connection between the pair of positions in the subgoal graph that leads closest to a global solution is tested next. This best pair is calculated based on all the knowledge of the subgoal graph that is available at that time. If, for example, a point that was unconnected before, gets connected to the tree – and thus to the start – the connection to the goal is tested immediately.

PRACTICAL RESULTS

The experiments shown in this chapter were performed on Hewlett Packard Unix Workstations. The geometry simulation 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. More results of sequential planning are shown in [2].



(a) (b)

Figure 2. *MOBROB scenario (a) and PYRAMID scenario (b). See text for details.*

Sequential Planning in the Scenario MOBROB

This scenario (Figure 2 a) 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 subgoals and a maximum of $m = 4$ subgoals on one path. 5,000 tasks are 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 efficiency of the local planner, that can solve almost all tasks
- the local planner is used 1.16 times on an average.

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 where the global planner becomes necessary the planning times are usually below one second.

Parallel Planning in the Scenario PYRAMIDS

The way of parallelizing the Z^3-method by scheduling the local planning requires little communication, compared with the high computational efforts for the kinematic and geometric simulation. For this reason we chose a cluster of workstations as the development environment. In the shared file system, all computing nodes have access to the same data. The processes are controlled and interconnected through *PVM* (Parallel Virtual Machine [9]).

The pyramid scenario is created as an example where local planning with slide steps fails very often, due to the large number of obstacles very close to the robot. The task shown in Figure 2 (b) (start position: solid drown robot, goal: wire frame robot) requires a complete turn by 350 degrees in the workspace. The six dimensional *c-space* contains 37.5% free space. The simulation system needs an average time of 0.44 ms to check a position for collision. The sequential implementation solves this task in an average time of 31.2 sec (100 runs).

The parallel planner is started 100 times with $M = 50$ total subgoals and $m = 5$ maximum number of subgoals on one path. We use 30 workstations running the local planner in parallel, controlled by one master workstation, running the scheduling algorithm. The average planning time including all communication is 7.5 sec. Compared to the sequential algorithm, this is a speedup by the factor 4.

The results achieved with the parallel planner are preliminary. The scheduling strategy implemented now suffers one major drawback: the direct connection between start and goal is tested first in one computing node, and only if this fails, multiple instances of the local planner start working. It will increase the performance to start searching connections via subgoals immediately when the planning begins, as there are idle computing nodes available.

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, time being an additional degree of freedom. Some problems and objections, as well as ongoing and future work, are 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 [4].

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 can be done without increasing the planning 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], and an integration into parallel planning is straightforward.

In static environments the Z^3-method does 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. [5, 7], show better results for repeated planning in the same environment. In our opinion, the dominating criterium is the constant efficiency 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 [3], but the '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 randomized and parallizable 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*, Minneapolis, April 1996.
2. 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, Orlando, Florida, January 1996.
3. 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.
4. 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.
5. 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.
6. Bernhard Glavina. *Planung kollisionsfreier Bewegungen für Manipulatoren durch Kombination von zielgerichteter Suche und zufallsgesteuerter Zwischenzielerzeugung*. PhD thesis, Technische Universität München, February 1991.
7. 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.
8. Jean-Claude Latombe. *Robot Motion Planning*. Kluver Academic Publishers, 1991.
9. V. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency: Practice and Expirience*, 2(4), December 1990.