Motion Planning for Manipulators in Dynamically Changing Environments using Real-Time Mapping of Free Workspace

Mihai Pomârlan[1] and Ioan A. Şucan [2]

[1]Universitatea Politehnica Timisoara, facultatea ETC
[2]Willow Garage

***Abstract*—This paper introduces a method to efficiently compute global motion plans for robotic manipulators in dynamically changing environments. An offline computation step is used to construct a sparse roadmap to approximate the configuration space of the manipulator in an empty environment. When the robot is running, a representation of the environment to keep track of the robot's free workspace is maintained as sensor updates are received. The maintained representation of the free workspace is used in conjunction with the data computed offline to quickly compute good quality global motion plans.**

***Index Terms*—robot motion planning, sparse roadmaps, dynamic environments**

I. Introduction

Motion planning is the problem of finding a continuous path from start to goal, under a specified set of constraints [1], [2]. Solving this problem has applications in robotics, but also in other domains (e.g., graphics, computational biology [3]).

Different sets of constraints need to be used in the computation of motion plans depending on the system considered (e.g., robots moving at higher velocities likely consider dynamic constraints, robots transporting objects may need to maintain particular orientations for the objects). This paper considers the case of manipulators operating in dynamically changing environments. We wish to obtain collision free paths as quickly as possible. We assume the manipulator can be controlled such that motions specified as sequences of waypoints can be followed. This is a reasonable assumption since most manipulators allow this form of control, and making such an assumption allows for planning in the configuration space of the robot – computed motion plans are in fact sequences of waypoints, each waypoint being a configuration of the manipulator (sometimes referred to as "path planning" or planning under geometric constraints [1], [2]).

The difficulty of the problem considered here lies in the fact that operating in dynamically changing environments potentially requires frequent re-computation of paths. For example, when operating around humans, or even in industrial contexts, the speed of reaction of the robot is very important. In this work we do not consider the issue of safety explicitly – we assume a safety maneuver is executed when appropriate and a new motion planning request is made. Thus we focus our work on computing this new plan as quickly as possible. Furthermore, we attempt to produce shorter solutions directly from the motion planner, without requiring subsequent path shortening heuristics. Because manipulators are usually relatively high-dimensional systems (typically 6 or 7 degrees of freedom (DOF)), approaches based on deterministic search are typically slow [4]. Sampling-based motion planners have been shown to quickly compute motion plans for manipulators, but even in the best scenarios, only a few motion plans can be computed per second [4].

We develop a sampling-based method to keep track of the free part of the robot's workspace as new sensor data is received and efficiently use this data, in conjunction with data structures computed offline. The resulting approach is faster than typical sampling-based planning algorithms by a factor of two to three (and sometimes even higher).

This paper is organized as follows. We discuss some related work in Section II, and we describe our approach in Section III. Experimental results are shown in Section IV. Conclusions and future work are in Section V.

II. Related Work

There are a number of approaches addressing the problem of planning in dynamically changing environments. From a control perspective, Haddadin et al. [5] show a method that can avoid local collisions for a manipulator in real-time (at 0.5 kHz). This method is very fast, but it is not intended for global motion planning. In the context of sampling-based motion planning, the idea of replanning (e.g., [6], [7]) can be used in the context of dynamically changing environments. Between replanning steps (which compute only a short motion plans) updates to the environment can be incorporated. Elastic roadmaps [8] can also account for limited changes in the environment.

In this work we use the idea of constructing an approximation of the manipulator's configuration space, which has been also discussed in previous work. Zacharias et al. [9] used a previously computed set of robot configurations in an RRT [10] algorithm to produce paths that appear more human. The idea of using an offline-computed roadmap to approximate constraint manifolds has also been explored [11].

III. Method

In short, the proposed method consists of an offline step and an online step. The offline step computes a roadmap [12]

[1]mihai.pomirlan@etc.upt.ro
[2]isucan@willowgarage.com

This work was partially supported by the strategic grant POSDRU 107/1.5/S/77265, inside POSDRU Romania 2007-2013 co-financed by the European Social Fund Investing in People.

approximating the configuration space of the manipulator, assuming an empty environment. The online step answers motion planning queries by efficiently using the roadmap computed in the offline step and a continually maintained representation of the free workspace.

A. Offline computation

The offline computation consists of constructing a roadmap $\mathscr{R} = (\mathscr{V}, \mathscr{E})$ in the configuration space of the manipulator, assuming an empty environment, but avoiding self-collisions. Let $\mathscr{V}$ be the set of vertices making up the roadmap and $\mathscr{E}$ be the edges in the roadmap. This step can use PRM [12] to grow the roadmap, but to achieve a low memory footprint and obtain good quality motion plans, we use sparse roadmaps [13] constructed with the OMPL[14] SPARS2 package. Figure 1 shows the positions of the end effector, corresponding to each vertex in the roadmap, as pink markers around the robot.

The graph is assumed undirected, as all movements of a manipulator are reversible. For each edge we associate a cost value, which is the cost a path must pay to use that edge. This cost could depend on the distance between the incident vertices, or on some other, problem specific metric like energy consumption. Each vertex also has a cost value associated to it, which is the cost a path must pay to go through that vertex. All vertices start with cost 0.

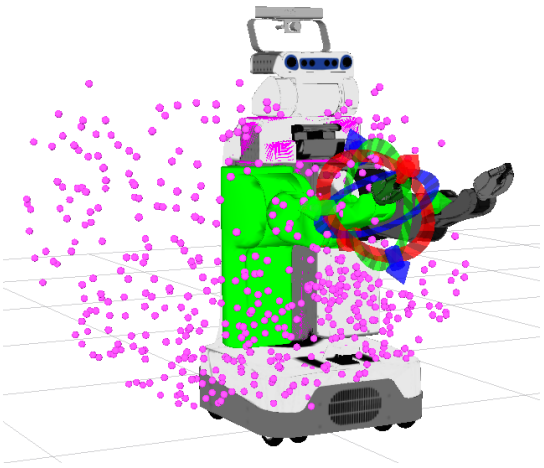

Fig. 1. Roadmap vertices for one manipulator arm. The end effector positions corresponding to the vertices are shown as purple markers.

B. Online computation

While the robot is running, we maintain a representation of the workspace around the manipulator using a 3D sensor. In our work we used a Kinect^{TM} sensor, but any source of point cloud data could have been used. We use the Octomap[15] library for maintaining an octree representation of the environment.

Octomap stores a probability of occupancy for each cell the environment. Cells below a specified threshold are considered free and cells above a specified threshold are considered occupied. It allows quick checks to see whether a particular point is in an occupied cell or not. However, a full collision check with a robot arm configuration is fairly expensive (around one millisecond on the hardware we used), and certainly impractical for a naive approach that would recheck the entire roadmap at every planning request.

Therefore we decide to have the planner itself maintain a representation of what it "believes" to be occupied or free space, and it will update this representation as required by the problems it is given to solve. In this paper, we have the planner update its internal representation of the free space only while it solves planning queries, and it does so by adjusting vertex costs based on how close they are to vertices discovered to be invalid while checking candidate solutions. However, the vertex costs can also be updated by a background thread that periodically checks validity of random vertices based on new octomap data, even when planning requests are not processed. The details of the cost update process are detailed in the next section.

C. Processing a motion planning request

The planner we use is inspired by LazyPRM on the sparse roadmap. For completeness, we describe LazyPRM here, then give a reason as to why by itself it is not enough to achieve the speed-ups we want, and present a way to improve LazyPRM's performance.

When given a planning query- a pair of start and goal states- LazyPRM adds temporary vertices to the roadmap and links them to the closest k neighbors (we use a value of 40 here). A shortest path algorithm produces a candidate path in the roadmap. All vertices along the proposed path get checked for validity, and then all the edges. If all vertices and edges are free of collision, the path candidate is the solution.

Suppose the candidate path is not valid however. The first invalid point found, be it a vertex or a point along an edge, stops the validation process; the candidate path fails, the offending edge or vertex is marked as unusable, and a new graph search will be performed, seeking another path candidate while avoiding unusable vertices or edges.

One can say LazyPRM starts working on a graph G, obtained from the roadmap together with the start and goal vertices, and first produces the shortest path inside this graph linking start to goal. Should the path not be valid, it produces a new graph G', the result of removing the most recently detected invalid part from G, and then finds the best path inside this new graph etc.

One notices LazyPRM has a focus on optimizing path length (with the optimization restricted to paths already available in the graph). This creates a problem when speed is desired: the second best path, usually, isn't too different from the best path, ie., it often passes through the same regions, or nearby. So if one obstacle invalidates the best path, it is often the case it invalidates the second path too. As a consequence, in trial runs we found LazyPRM would try several short paths, before eventually finding a feasible one.

Therefore we need a way to push LazyPRM away from seeking the shortest path, and away from likely obstacles. After

finding an invalid roadmap element, but before attempting a new graph search, all (usable) vertices in the roadmap, except the temporary vertices start and goal, receive a cost bump, which depends on the square of their distance to the invalid point detected. Initially, the cost associated with each roadmap vertex is 0; a cost bump will increment the cost of vertex **x** by:

$$c_b(\mathbf{x},\mathbf{p}) = \frac{q}{1+\left(\frac{\|\mathbf{x}-\mathbf{p}\|}{r}\right)^2}$$

where **p** is the point where a collision was detected, r is a radius parameter, and q is a maximum penalty parameter.

The roadmap used here is sufficiently small (357 vertices) and the square of distance sufficiently fast, so a cost bump update takes a negligible amount of time compared to, for example, a collision check. One could of course limit the number of vertices updated using some data structure for nearness queries.

Vertex cost bumps therefore function as a trade-off between attempting short paths, and keeping clear of short paths that have proven to fail. They can also function as a representation of occupied space internal to the planner, as long as this representation can be updated both ways: increase cost when it appears a vertex may be close to an obstacle, decrease it when it appears close to a free zone. Cost reduction would be analogous to the cost bump: when a vertex is found valid, its cost becomes 0, and the costs of its neighbors are reduced depending on distance similar to the cost bump, where we limit cost reduction to never make a vertex cost less than 0.

Since there is a difference in how node costs and usability flags behave, they should be handled differently. A high cost vertex may still be considered by the graph search, whereas an unusable one will not be. So while we can keep vertex costs from one planner run to another, as a persistent representation of the environment, usability flags must be reset, or else we risk losing more and more vertices and irreversibly impoverishing the roadmap.

Conditions for cost bump or reduction are straightforward to track while answering a planning query (a vertex or edge fails a check for a cost bump; a valid check result causes cost reduction). These conditions could also be tracked on longer stretches of time, for example interleaved among sensor update operations in a lower priority thread. This thread would select random vertices from the roadmap, run validity checks on them, and update costs of nearby vertices as necessary. Vertex costs could also decay with time.

Here we employ two different simple strategies. One is to reset all 'unusable' flags and vertex costs once the planner terminated its run (be that because a plan was found or because of timeouts), so each planning run starts from the same initial vertex costs. The other is to also have a 'cost unbump' function: when a vertex is found to be valid, it and its neighbors have their costs reduced. The cost reduction function has a similar shape to the cost bump, but cannot get the cost of a vertex below 0.

Algorithm 1 LPRM with cost bumps (G, q_{start}, q_{goal})

```
startNear ← NearestNeighbors(G, q_start, r)
goalNear ← NearestNeighbors(G, q_goal, r)
for q ∈ startNear do
   addEdge(G, q_start, q, |q_start − q|)
for q ∈ goalNear do
   addEdge(G, q, q_goal, |q_goal − q|)
G' ← G
while time remaining do
   shortestPath ← computeShortestPath(q_start, q_goal, G')
   if pathIsValid(shortestPath, collisionState) then
      return shortestPath
   else
      G' ← G' − {collisionState}
      for each q ∈ G' do
         q.cost ← q.cost + c_b(q, collisionState)
return no solution
```

IV. Experimental Evaluation

We used The Open Motion Planning Library (OMPL)[14] and MoveIt! [16] for the implementation.

For a first set of tests, we ran our planner and RRTConnect on a set of planning queries and recorded execution times and path lengths. The queries are planning problems requiring the manipulator to move around a table (taking the end effector from beneath the table to above it, for example) as well as around objects on that table. Our planner was run for 30 times for each problem. Every planner run started from zero vertex costs, and the planner had to rediscover the environment. Because its results show more variation, RRTConnect was run 100 times for each problem. Averages and standard deviations of the results from these runs are available in tables I and II for execution times and path lengths respectively. Box plots are given in figures 3 and 4.

Fig. 2. Robot planning environment

As the table and plots reveal, the proposed planner is

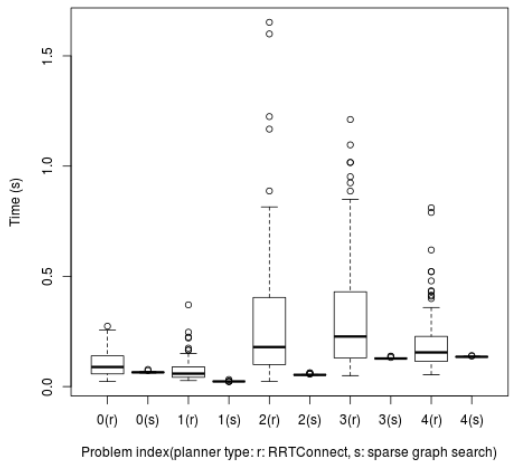

Fig. 3. Boxplots: planning times

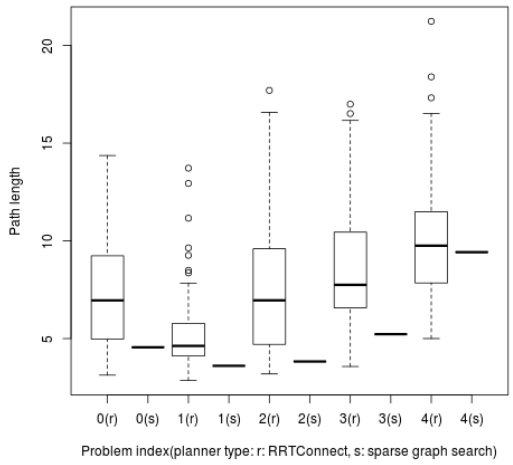

Fig. 4. Boxplots: path lengths

TABLE I
AVERAGE AND STANDARD DEVIATION FOR EXECUTION TIMES

Problem	RRTConnect		New method	
	Avg time(s)	StD time(s)	Avg time(s)	StD time(s)
0	0.102	0.057	0.065	0.003
1	0.075	0.053	0.024	0.002
2	0.292	0.309	0.054	0.003
3	0.331	0.270	0.128	0.003
4	0.201	0.143	0.136	0.002

TABLE II
AVERAGE AND STANDARD DEVIATION FOR PATH LENGTHS

Problem	RRTConnect		New method	
	Avg length	StD length	Avg length	StD length
0	7.314	2.895	4.553	0.000
1	5.205	1.900	3.606	0.000
2	7.597	3.353	3.826	0.000
3	8.626	3.238	5.224	0.000
4	9.976	3.068	9.420	0.000

capable of finding better quality paths faster than RRTConnect, sometimes twice as fast or better. The performance of the planner is dependent of course on the sparse roadmap used, which should be small enough to enable fast queries, but large enough to capture manipulator movements that would allow it to move gracefully in a cluttered environment.

For a second set of tests, we keep the vertex cost values from one planning problem to another. We run our planner 30 times for each problem, and each planner run starts from the same initial vertex costs. However, the vertex costs at the end of the last run for a problem will become the initial vertex costs for all runs of the next planner problem. To compare, we use RRTConnect, which we run for 100 times for each problem. We again collect averages and standard deviations of planning time and path length. Boxplots for planning time are shown in figure 5; statistics for planning time and path length are also shown in tables III and IV respectively.

TABLE III
AVERAGE AND STANDARD DEVIATION FOR EXECUTION TIMES

Problem	RRTConnect		New method	
	Avg time(s)	StD time(s)	Avg time(s)	StD time(s)
0	0.174	0.072	0.082	0.001
1	0.242	0.200	0.097	0.002
2	0.157	0.075	0.127	0.001
3	0.165	0.079	0.252	0.012
4	0.173	0.080	0.142	0.003
5	0.209	0.198	0.054	0.002
6	0.197	0.183	0.136	0.002
7	0.181	0.078	0.103	0.002
8	0.163	0.069	0.083	0.002
9	0.097	0.092	0.048	0.002
10	0.207	0.169	0.158	0.001
11	0.284	0.180	0.158	0.002

TABLE IV
AVERAGE AND STANDARD DEVIATION FOR PATH LENGTHS

Problem	RRTConnect		New method	
	Avg length	StD length	Avg length	StD length
0	10.644	2.563	6.861	0.000
1	7.910	3.321	6.552	0.000
2	10.253	2.524	7.959	0.000
3	10.461	2.244	10.008	0.000
4	10.984	2.808	10.008	0.000
5	6.361	2.887	2.938	0.000
6	7.250	3.731	3.525	0.000
7	11.350	2.564	9.796	0.000
8	10.876	2.505	5.725	0.000
9	5.599	2.375	4.212	0.000
10	6.977	2.534	9.977	0.000
11	10.406	3.342	7.777	0.000

Again we can see our planner is usually faster than the RRTConnect average and median. One exception is problem 3. Problem 4 is the same start/goal state pair, and our planner is now faster, because vertex costs help steer the planner away from some dead ends. Also the environment changes between problems 5 and 6, but our planner re-adapts costs quickly and maintains its efficiency.

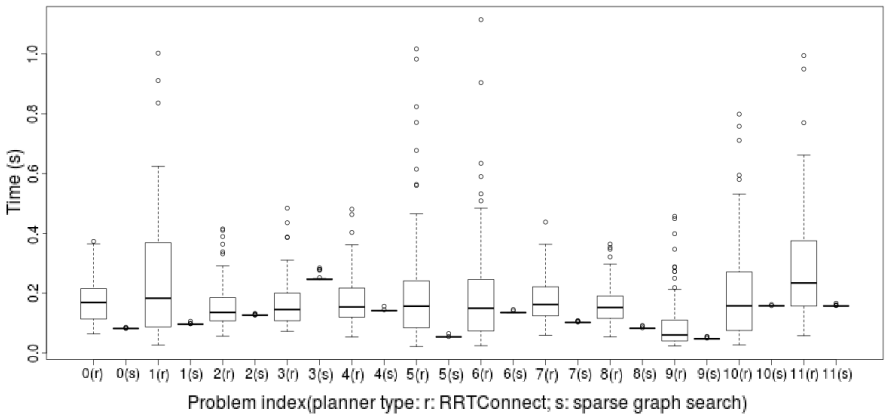

Fig. 5. Planning times with persistent costs

V. Conclusions

The proposed heuristic of adjusting vertex costs while planning proved a promising way to obtain good quality plans and fast planning times. Having a good precomputed roadmap however is key; the roadmap needs to be small enough to be quick to query, yet rich enough to capture enough variety of behavior for the robot. It may be useful, as future work, to investigate other procedures for roadmap generation, not just sparse planners; for example, some other methods that explicitly take into account the geometry of the configuration space.

While the method presented here is often faster, it can however fail to return a plan. This happens if enough obstacles appear to disconnect the precomputed roadmap, and the start and goal get connected to different components. It is possible to detect when this happens, however, and in such a case, one can employ another planner, like RRTConnect, as fallback. Alternatively, one could run our planner and RRTConnect in parallel, and return the first solution.

Presently the cost bump/unbump policy only affects vertices. Edges passing near to an invalid vertex, but whose endpoints are far away, are not affected. Also, often it is edges that are found invalid. We used the first invalid point on an edge as a dummy vertex to bump vertex costs around, however no equivalent heuristic exists for cost unbump when an edge is found valid in its entirety. Some adjustment to the heuristic to better account for edges will be investigated.

References

[1] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, June 2005.

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

[3] J.-C. Latombe, "Motion planning: A journey of robots, molecules, digital actors, and other artifacts," *Intl. Journal of Robotics Research*, vol. 18, no. 11, pp. 1119–1128, 1999.

[4] B. Cohen, I. A. Şucan, and S. Chitta, "A generic infrastructure for benchmarking motion planners," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, Portugal, October 2012, pp. 589–595.

[5] S. Haddadin, R. Belder, and A. Albu-Schäffer, "Dynamic motion planning for robots in partially unknown environments," in *Proceedings of the 18th IFAC World Congress*, vol. 18, 2011, pp. 6842–6850.

[6] J. v. d. Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *IEEE Intl. Conference on Robotics and Automation*, Orlando, Florida, May 2006.

[7] K. E. Bekris and L. E. Kavraki, "Greedy but safe replanning under kinodynamic constraints," in *IEEE Intl. Conference on Robotics and Automation*, Rome, Italy, April 2007, pp. 704–710.

[8] Y. Yang and O. Brock, "Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation," in *Robotics: Science and Systems*, Philadelphia, Pennsylvania, August 2006.

[9] F. Zacharias, C. Schlette, F. Schmidt, C. Borst, J. Rossmann, and G. Hirzinger, "Making planned paths look more human-like in humanoid robot manipulation planning," in *IEEE Intl. Conference on Robotics and Automation*, Shanghai, May 2011, pp. 1192–1198.

[10] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Dept., Iowa State University, Tech. Rep. 11, 1998.

[11] I. A. Şucan and S. Chitta, "Motion planning with constraints using configuration space approximations," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, Portugal, October 2012, pp. 1904–1910.

[12] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, August 1996.

[13] A. Dobson, A. Krontiris, and K. E. Bekris, "Sparse roadmap spanners," in *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, June 2012.

[14] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.

[15] K. M. Wurm and A. Hornung, *Octomap: 3D occupancy grid mapping based on octrees*, 2010, software available at http://octomap.sf.net.

[16] I. A. Şucan and S. Chitta, "MoveIt!" [Online]. Available: http://moveit.ros.org