Improving Efficiency of Intricate Manipulation Planning through Mapping of Grasp Feasibility Zones

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

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

***Abstract*— One intuitive approach for planning robotic manipulation tasks is to first compute a plan for the manipulated object, as if capable to move on its own, then use the obtained plan to find a sequence of arm maneuvers to take the object along the computed plan. Motion planning queries would be performed in relatively low-dimensional configuration spaces, rather than the full configuration space of a robot's arms and grippers, resulting in a more efficient planning method. However, having a plan for the manipulated object does not guarantee there also exists a feasible sequence of maneuvers for the robot to do the manipulation. Knowing different possible grasps on the object increases the robots chance to find a sequence of grasp switches and maneuvers, but makes the search more time consuming as the grasps need to be tested for usefulness. In this paper, we develop a multi-level architecture for complex manipulation planning of rigid bodies which uses communication between the two levels, one for planning the manipulated object motion, the other to plan for the arms, to improve both the robustness and efficiency of the method. We store grasp zones in the configuration space of the manipulated object as regions where a given grasp seems promising. We use grasp zones to guide searching for grasp switching maneuvers, and to avoid regions of configuration space where few good grasps exist.**

***Index Terms*— manipulation planning, motion and path planning, sparse roadmaps, human assisted planning**

I. Introduction

One of the tasks a robot has to do, if operating in a human environment, is to tackle manipulation problems that are second nature to humans. Objects often need to be entangled with, or disentangled from, one another; putting a key or removing it from a key chain, unhooking a hammer from a tool rack, or tying a knot would be some examples. These problems often require narrow features for their solutions and grasp changes during object manipulation.

One could treat such problems as motion planning queries in a configuration space of all of a robot's arms and grippers, as well as any remaining degrees of freedom of the manipulated object. Such a configuration space would however be very high-dimensional. A more efficient approach seems to be treating the manipulated object as if it were capable to move on its own, and pose the manipulation problem as a motion planning query in this smaller configuration space. Once a solution trajectory for the manipulated object is found, the problem becomes one of finding a sequence of arm movements and grasp changes with which the robot can take the manipulated object along the solution trajectory. However, this approach also has a few problems, as we discuss below.

A solution trajectory for the manipulated object may be hard to find, because it may use narrow passages in the configuration space. Sampling-based algorithms [1] are used for motion planning because they are typically faster than other methods, but have difficulty handling small-volume features [2]–[4].

Assuming a solution trajectory for the manipulated object is found, the robot may not be able to take the object along it. The solution trajectory may need the robot to reach through obstacles, or to bring its grippers to infeasible poses. While it is more efficient than exploring a high-dimensional configuration space, a two-level approach such as this is not probabilistically complete.

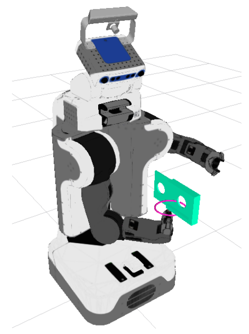
Fig. 1. An example manipulation problem class: the robot must maneuver the ring piece around the fixed card. Narrow features and the necessity of grasp changes make such problems challenging.

Also, the robot must find what grasps to use while following the solution trajectory. The more grasps the robot knows, the more capable it should be to solve manipulation problems. However, if too many grasps are known, but no way to guide the search through them, a large amount of time will be spent testing many sequences of grasps. Some way to prioritize which grasps are tested is needed, ideally one which will typically suggest good grasps, so that a sequence to follow the solution trajectory is found quickly.

In this paper, we build a two-level manipulation planning approach to address the above problems. At the first level, we store a roadmap for the manipulated object configuration space, constructed by a multi-query planner with human assistance to identify narrow passages and useful trajectory segments. We store grasp suggestion information on roadmap vertices, to guide the grasp selection process. At the second level of planning we search for a sequence of grasps and arm movements to take the manipulated object along the solution

[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.

trajectory. We update cost information in the roadmap based on grasp selection results. The cost updates steer solution trajectories away from regions where the robot has few grasp options available in its environment. The second level of planning also decides whether to call on the first for a new solution trajectory if the current one appears too difficult to follow. We implement the planning method using the MoveIt! package [5]. We test our method on a simple disentanglement puzzle that showcases some features of manipulation problems. The puzzle we consider in this paper involves two rigid objects, one of which is movable; several planning problems can be defined with the two objects. Their geometry is such that narrow passages exist in the puzzle's configuration space; also, manipulations will often require grasp switches. We show our method is capable to find solutions robustly and efficiently. We describe the method in Section III. We give some experimental results and analysis in Sections IV and V.

II. Related Work

The planners employed here are sampling-based, in the vein of PRM [6], [7] and its variants, sparse roadmaps constructed by the visibility heuristic [8] with useful cycles [9]. Guiding planning in a high dimensional space by using a lower dimensional plan has appeared before in work such as [10] or [11]–[13]. In [10], a discrete decomposition of the workspace is the lower dimensional guide; [11]–[13] treat motion planning under constraints to remain in submanifolds of the workspace that are either known beforehand (e.g., [11], [12]) or identified on-line by principal component analysis on the expansion results of a single-query planner [13]. In our work we use a continuous guide, the solution trajectory for the manipulated object. We do not consider constrained motion here, although our approach could use specialized sampling to handle constraints.

Human assistance for planning operations has been used as a way to improve planning capabilities [14]. Such research aims to have a human user demonstrate a task, so the robot could then perform it [15]. Autonomy is not the focus of such research, the goal is to find new and more user-friendly ways to communicate a program to a robot.

Task planning also distinguishes between two levels: symbolic, encoding the logical aspects of a task, and geometric, describing the world the robot works in. The symbolic level reasons about subtasks, preconditions and effects, and searches for a sequence of subtasks to solve a given problem. A sequence of subtasks may be logically feasible (no subtask is started before its logical preconditions are met), but geometrically infeasible (there is no solution at the geometric level). Methods for communication between the two levels have been proposed.

"Manipulation graphs" [16], [17] are one approach. Subspaces of the robot configuration space are defined based on the grasp on the manipulated object. A subspace in which the robot does not grasp anything is also considered. Sampling-based planners generate roadmaps for the subspaces, and one can go from one grasp subspace to another by passing through the no-grasp subspace. The manipulation graph is then a map of known transitions between the subspaces.

Extensions of the manipulation graph concept exist. In [18], constraint satisfaction algorithms on the geometric level specify destination configurations for subtasks. In [19], the problem of infeasible subtasks is handled by concurrent PRM searches on several subtasks, and the found feasible solutions are used. In [20], Task Motion Multigraphs capture the several possibilities a redundant manipulator has to achieve a goal.

The method presented here does not maintain several roadmaps for different grasp states. We keep only one roadmap of the configuration space of the manipulated object, but with some extra information besides the configurations at the roadmap vertices which is used to guide grasp selection on the second level, of arm planning, and to steer graph searches away from areas where the robot has few or no grasps feasible.

III. Algorithm description

A. Constructing the Roadmap

The problems considered here are about entangling/disentangling two rigid objects to/from one another. We consider one object as fixed in the environment; the other is movable. We use a multi-query, sparse roadmap planner to construct a roadmap for configurations of the movable object around the fixed one. Initially, we consider the environment empty except for the two objects.

We grow the roadmap for the rigid object pair by issuing several planning queries. The start and goal states of these queries are added to the roadmap if they pass a visibility heurisic (either they cannot be connected to other samples in the roadmap, or allow connections between different connected components, or allow useful cycles [9]). Roadmap construction benefits from input of a human operator who issues the queries, and who can identify some useful configurations and narrow passages easily. The roadmap construction is illustrated in Algorithm 1, where r is a connection radius parameter and $ULoop$ is a function taking a configuration as input and returning true if there is a point on an edge between its two closest neighbors that cannot be reached from this configuration [9].

Algorithm 1 GrowRoadmap (G, q_{start}, q_{goal})

```
$startNear \leftarrow$ NearestNeighbors$(G, q_{start}, r)$
$goalNear \leftarrow$ NearestNeighbors$(G, q_{goal}, r)$
$nearVerts \leftarrow startNear \cup goalNear$
$cCs \leftarrow GetConnectedComponents(nearVerts)$
if $|cCs| \neq 1$ or $ULoop(q_{start})$ or $ULoop(q_{goal})$ then
    $G \leftarrow G \cup \{q_{start}, q_{goal}\}$
    for $q \in startNear$ do
        addEdge$(G, q_{start}, q, |q_{start} - q|)$
    for $q \in goalNear$ do
        addEdge$(G, q, q_{goal}, |q_{goal} - q|)$
    RunPlanner($G$, $q_{start}$, $q_{goal}$)
```

Eventually, as the number of samples grows after queries from the human operator, the connected components will merge into one maximal component. For the example problem used in this paper, we prepared a roadmap with 270 vertices.

B. Grasp definitions and computing grasp zones

We assume there is a finite collection of known grasps on the manipulated object. The grasps could be provided by a grasp planner, or defined by the human programmer; in this paper, we took the second approach for our test problem. Two grasp examples are shown in Figure 2. In general, a grasp defines a rigid body transformation which relates the pose of the movable object to the pose of the gripper on the robot, and the values for the gripper joint coordinates that describe the gripper shape in the grasp. The grasp collection is constructed off-line.

For a grasp to be feasible on the manipulated object, the arm should not collide with obstacles, as well as have a solution to the inverse kinematics problem: given the pose of the gripper, find a set of joint values for the arm. For a grasp to be feasible along a trajectory, a third condition should also hold: the inverse kinematics (IK) solutions should not jump discontinuously.

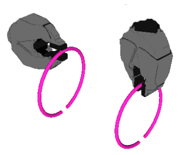

Fig. 2. An "aligned" grasp (left) and a "side" grasp (right) on the ring piece of our example problem. Several points along the ring are allowed for grasping, and for each, grasps can be "aligned" or "side".

At some point during manipulation planning, we will have to select what grasp to use from the feasible ones. The more grasps the robot knows, the more capable it is, but having more choice also implies more planning time unless there is some way to rank the choices. Preferably, the ranking should be such that high ranked choices tend to be grasps that are feasible for long stretches of the solution trajectory for the manipulated object.

To provide such a ranking, we study what grasps would be good in a given region of the manipulated object's configuration space. For each vertex **x** in the roadmap, and each known grasp **g**, we define a grasp zone for **g** around **x** as z(**x**, **g**): the minimum among the maximum distances one can use the grasp **g** while moving the object on edges starting from **x**. If the grasp **g** is infeasible at **x**, the grasp zone is 0. This processing is done off-line, after the roadmap construction phase.

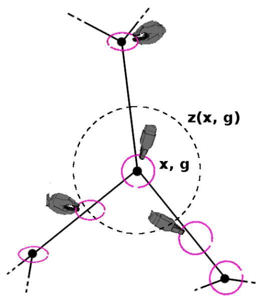

Fig. 3. Getting a grasp zone for a grasp **g** around a vertex **x** in the object roadmap: follow the neighboring edges using **g**. The shortest distance before it becomes infeasible is the grasp zone.

During the planning stage, suppose we are at some point in the solution trajectory for the manipulated object described by a configuration **y**, and we need to change the grasp because we cannot use it to follow the trajectory anymore. The solution trajectory may be post-processed after planning (MoveIt! [5] includes a path simplification step, for example), so we do not assume that **y** is a vertex or on an edge in our roadmap, but we can run a k-nearest query between it and roadmap vertices (in our case, we used $k = 5$); let $\mathbf{N}_k$ be that set of vertices. From each of the k-nearest neighbors, we return a list of grasp suggestions where, from each vertex **x** and each grasp **g** we define a suggestion strength:

$$s(\mathbf{x},\mathbf{g},\mathbf{y}) = \frac{z(\mathbf{x},\mathbf{g})}{(1+|\mathbf{x}-\mathbf{y}|)}$$

where $dist(\mathbf{x},\mathbf{y})$ is the distance between the two configurations.

For each grasp, we store the largest suggestion strength value from the k-nearest neighbors.

$$s(\mathbf{g},\mathbf{y}) = max_{\mathbf{x}\in\mathbf{N}_k}(s(\mathbf{x},\mathbf{g},\mathbf{y}))$$

We sort the grasps in descending order of suggestion strength; this is the order in which we will try them at **y**.

C. Planning for the Manipulated Object

When solving a manipulation problem of the dis/entangling a rigid body from/to another type, our first step is to obtain a solution candidate trajectory from the roadmap for the rigid object pair. The method is lazy PRM [21]: we run a shortest path algorithm, and check vertices and edges along the proposed solution. If some are invalid, we mark them as unusable by flipping a validity flag and try again. After the planner terminates, all validity flags are reset. The planning environment is the environment the robot has to work in, with whatever obstacles are present. The robot itself is not considered an obstacle, because we assume we can get it out of the way if needed.

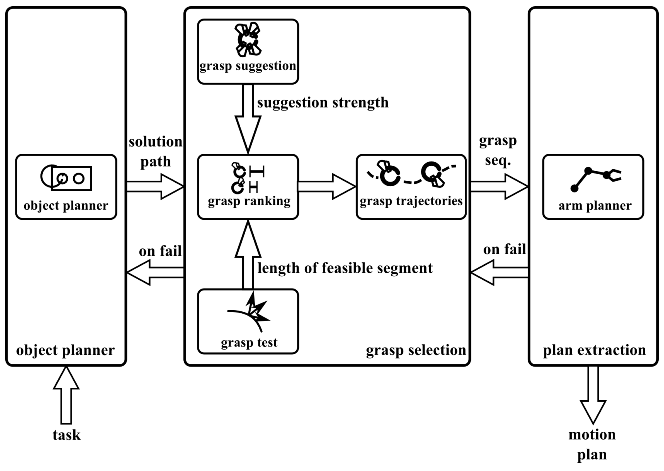

Fig. 4. Planning for a manipulation task. First, a solution path candidate is obtained for the manipulated object. A sequence of grasps to follow the solution candidate is then sought. Grasps are first ranked by suggestion strength (a fast operation) before testing for length of feasible segment along solution candidate (a better, but time consuming, measure of grasp quality). The grasp sequence is then converted to arm planning queries. Failure conditions allow backtracking between the three levels to attempt new grasp sequences or object solution path candidates.

For the graph search algorithm, we use edge weights related to lengths in the configuration space. We also associate a cost value for each vertex. Initially, all vertices in the roadmap have cost 0. Costs are not reset after the planner terminates, because they accumulate information about the current obstacles in the environment.

One way for costs to increase is a "cost bump" function to speed up plan searches by steering them away from invalid zones of configuration space. We apply a cost bump function whenever we find an invalid vertex on solution candidates; the rationale is, if a vertex is invalid (because some obstacle has appeared in the environment), then vertices near it are likely invalid too. Assuming some configuration $\mathbf{y}$ in a plan candidate is invalid, we increase the cost of vertex $\mathbf{x}$ using the formula:

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

where r is a radius parameter and q is a maximum penalty parameter. Since our roadmap is small, and the cost bump can be evaluated quickly for each vertex, we compute and apply cost adjustments to all vertices in the roadmap. A nearest neighbors query can also be used to filter its range.

Vertex costs are a way for the planner to approximate the free space; or, find where obstacles may be present and what regions to avoid. Since obstacles may move or disappear from the environment, we also reduce costs around vertices in a path candidate that is valid. The cost reduction function is the opposite of the cost bump, but is restricted in that it cannot take the cost of a vertex below 0.

There is also another type of event that triggers cost un/bumps for vertices on the roadmap, which is described in the next section.

D. Grasp Selection and Planning for the Arms

Once the planner has a candidate solution trajectory for the manipulated object, it looks for sequences of arm movements that will take the object along the solution. In particular, it looks for a sequence of grasps on the object. When one arm cannot maintain its current grasp on the object and follow the solution trajectory (either because of obstacles or inverse kinematics having no or discontinuous solutions), a grasp switch happens: the other arm grabs the object, the first arm releases it, and following the solution trajectory resumes.

A heuristic we found works well to keep the number of grasps small, and shortens the planning time, is to pick the grasp which allows the longest continuation along the solution trajectory. However, if many grasps are known, testing all may be inefficient. So we limit the set of grasps tested to the first m (we use $m = 8$) in the rank of suggestion strength $s(\mathbf{g},\mathbf{y})$, where $\mathbf{y}$ is the current configuration of the manipulated object.

Grasp selection is then two steps: (1) we rank grasps according to suggestion strength, an easy function to compute, and retain the first m; (2), we rank those m according to how long they can be used while following the solution trajectory. This second step is more time consuming, which is why we avoid performing it for many grasps. If the top-ranked grasps according to suggestion strength perform well, we will use those; if they fail to continue on the trajectory, we inspect other grasps.

It may be that at some state $\mathbf{y}$ there is no known way to grasp the object with the other arm. This is a dead end: we cannot continue with the current arm, and we cannot use the other. When a dead end happens, the planner backtracks along the trajectory to the previous moment where a grasp selection was needed, and picks another feasible grasp.

A dead end occuring tells us something else too: the object's solution trajectory passes through an area of configuration space where the robot has few grasping options. Maybe some sequence of grasps exists, but the necessity to backtrack will increase the planning time; and there may be no way for the robot to follow that solution trajectory anyway. We will then want to avoid that region of configuration space in the future, and we apply a cost bump to the object's roadmap, centered on the point where the dead end occurred.

Since dead ends signal a difficult area of configuration space, it may be the current solution trajectory is unfeasible for the robot. Therefore, after several dead ends happen (we use 5 as a limit), the planner searches for another solution trajectory for the manipulated object, and attempts to follow it instead. Because cost bumps steer search away from the dead end region, a new solution trajectory is generated. In principle, there can be parallel threads following several solution trajectories, but we did not take that approach here. Each new planning attempt starts fresh, on a new solution trajectory, with only vertex costs persisting from run to run to map the configuration space of the manipulated object in terms of it being free and reachable by the robot.

Since vertex costs are intended as a way to describe how hard regions of the configuration space are, there should also be a way to reduce costs, or "reward" regions that have good solutions. Therefore, if a solution path is followed in its entirety (and so allows a complete plan), we pick evenly spaced points on it and apply cost "unbumps" centered on those points. We do not assume any point in the trajectory is also in the roadmap; the trajectory may have been post-processed (for example, by MoveIt!'s path simplification), so it may no longer contain vertices or points along edges in the roadmap.

Arm motion planning is a time consuming operation if we need to plan many maneuvers to bring the arms to or away from the manipulated object. We delay planning for the arms and do so only when we have a sequence of feasible grasps that can take the manipulated object along the solution trajectory. We avoid motion planning queries on sequences of grasps that did not reach the end of the solution trajectory. In the next section, we will split the manipulation planning time into time for *grasp selection* (in which a complete and feasible sequence of grasps is found) and "*extraction*", the time spent for arm planning to perform the grasp switches in the sequence.

IV. Experimental Results

A. Test Problem Description

We consider here manipulation problems that involve moving one rigid object relative to another. To showcase the intricacies such problems present, we construct a simple example puzzle: a ring piece that must be moved around a card with two holes. A small arc misses from the ring, so it can slide along the card. Many planning queries can be defined with these two objects by specifying start and goal states for the ring; for example, unhooking it from one hole in the card and hooking it to the other. We assume the card to be a fixed obstacle, and the ring is assumed stable at the start and goal configurations. The ring however must be manipulated by the robot into following a solution trajectory; the robot is not allowed to drop the ring at any point on the solution trajectory except the start and goal states.

The robot we used for the simulation is the PR2 developed by Willow Garage, which has two 7-DOF arms. A ring manipulation problem will often require both of them. The planner is a modified OMPL plugin [22] included in the MoveIt! package [5]. Because our test problem has narrow features (the ring is very thin and the missing arc small), we configured the motion validity checker in OMPL to test many states along a trajectory. We ran simulations on a laptop with 3.8GB of RAM and an Intel®Core^{TM}i5-3210M CPU quadcore processor running at 2.50GHz.

B. Simulation Results

We give three motion planning problems: unhook the ring from one hole in the card and hook to the other (problem "change"); take the ring out, flip it, then re-hook it to the same hole ("flip"); take the ring, initially hooked to both holes, out, flip it, then re-hook the two holes ("dhook"). The start and end configurations for each problem are shown in Figure 5 a), b), c). First, we attempted to solve the planning problem for just the pair of rigid bodies (fixed card, movable ring; fewer dimensions than the configuration space of two arms with grippers) using RRTConnect [23]; we found that several minutes of searching did not provide solutions.

Next, we ran each problem five times with the method proposed here; in this test we do not store vertex costs from one run to another. To have a baseline of planning times, we first run our method with grasp suggestion disabled (equivalent to $m = \infty$), so whenever a grasp switch is needed all 24 known grasps for an arm are tested. In Table I we give averages and standard deviations of planning times, split for the stages of our method: plan for the manipulated object ("puzzle plan"), find a sequence of grasps ("grasp selection"), plan for the arms to bring them into the grasp positions or move them away as needed ("plan extraction"). We also give the number of grasp switches in the solutions; the number includes the first grasp the robot does on the object. We then repeat the test, with 5 runs for each of the planning problems, but now use grasp suggestions (with $m = 8$): we test the 8 grasps with the highest suggestion value, and only test more if these fail. The results are given in Table I. The results show the "grasp selection" process performs twice as fast compared to the $m = \infty$ case; fewer grasps are tested, but they tend to be good ones. All problems were solved, both by the $m = \infty$ and the $m = 8$ methods, with similar paths and grasp sequences. The other parts of the method ("puzzle plan": planning for the manipulated object, "plan extraction": planning for the arms) are not affected.

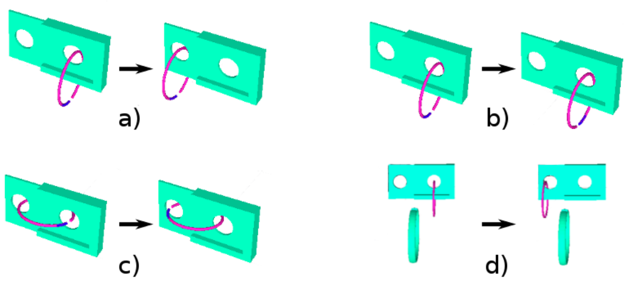

Fig. 5. A few example problems: a) "change", b) "flip", c) "dhook", d) an obstacle present. A thick ridge on the card prevents the ring from being simply rotated in place for the flip and dhook problems.

Next, we run the same planning problem ("change") several times, but we change the environment between some runs. For this test we keep vertex costs, to allow the planner to learn what parts of the object's configuration space to avoid. We also use grasp suggestions ($m = 8$). In Table II, we show the planning times of the various stages for each run, and the number of grasp switches on the resulting solution. Where several puzzle plans are attempted on the same problem run, we list all of them. Grasp selection is the time spent searching for grasps on all solution path candidates in a run.

All runs of this test use the same start/goal. Runs 1 and 2 are in an environment with only the robot, the card, and the ring. Both runs produce the same solution path. In run 3 we put an obstacle (see Figure 5 d)) to force the robot to seek a new solution; it attempts several candidates and settles on a solution with seven grasp switches. We do not move the obstacle for runs 4 to 6; however, they are much faster. The robot learned what regions of configuration space to avoid and settles on a five grasp solution path. We remove the obstacle for runs 7 to 9. The robot uses the same solution path as in runs 4 to 6, but now it can follow it with only four grasps switches. The robot does not revert to the path from runs 1 and 2; the cost of that region must decrease first, through other queries having endpoints there.

V. Conclusions and Future Work

We implement a method for intricate manipulation planning and test it on a class of problems with passages through narrow features and grasp switches. Our method of ranking grasps first by a quick suggestion heuristic is able to halve the time spent looking for a grasp sequence while also selecting good grasps and keeping the number of grasp switches small. Our method can also react to changes in the environment and learn which regions of configuration space should be avoided because they require awkward or impossible grasps, resulting in faster plans once such regions are known.

TABLE I

AVERAGES, STANDARD DEVIATION FOR PLAN TIMES, AND GRASP SWITCH COUNT FOR THE TEST PROBLEMS. GRASP SUGGESTIONS NOT USED.

	Problem	puzzle plan		grasp selection		plan extraction		grasp switches
		avg (s)	sdev (s)	avg (s)	sdev (s)	avg (s)	sdev (s)	
No grasp suggestion	change	0.4	0.04	6.5	0.16	5.24	0.59	2
	flip	0.52	0.02	15.63	0.26	16.22	0.97	7
	dhook	0.51	0.06	25.91	1.46	26.48	1.17	13
Grasp suggestion: first 8 of 24	change	0.37	0.02	3.25	0.07	5.5	0.42	2
	flip	0.49	0.01	6.71	0.12	15.89	1.0	7
	dhook	0.48	0.01	11.62	0.35	24.45	1.35	13

TABLE II

PLANNING TIMES AND GRASP SWITCHES FOR A "CHANGE HOLE" PROBLEM IN A CHANGING ENVIRONMENT

Problem run	1	2	3	4	5	6	7	8	9
puzzle plan (s)	0.33	0.33	0.37; 0.35; 0.37; 0.35; 0.35	0.39	0.39	0.43	0.40	0.42	0.40
grasp selection (s)	2.62	2.51	79.51	9.31	8.94	8.40	7.86	7.75	7.49
plan extraction (s)	5.04	5.03	13.46	9.31	8.94	8.40	7.86	7.75	7.49
grasp switches	2	2	7	5	5	5	4	4	4

We do not claim optimality of either path cost or number of grasp switches. Our heuristics are greedy, in an effort to obtain good quality plans fast, not perfect plans slower. We also do not claim probabilistic completeness.

The manipulated object roadmap determines the performance of the method, because it should contain enough trajectories to allow solution paths for many problems, as well as contain vertices for configurations that are not awkward to grasp. Constructing this roadmap is a time-consuming process requiring some human intervention to assist in finding narrow passages. In future work, we investigate possibilities to reuse and adapt the roadmap if the geometry of the objects changes slightly. Many manipulation tasks a human performs (such as unhooking tools from a rack or tying shoelaces) contain similarities, even if the geometries of the objects involved are not identical; we aim for the robot to be able to handle such classes of tasks, by giving it an adaptable roadmap for the manipulated objects.

REFERENCES

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

[2] D. Hsu, T. Jiang, J. Reif, and Z. Sun, "The bridge test for sampling narrow passages with probabilistic roadmap planners," in *IEEE Intl. Conference on Robotics and Automation*, Taipei, Taiwan, September 2003.

[3] M. Saha and J. Latombe, "Finding narrow passages with probabilistic roadmaps: The small-step retraction method," in *Intl. Conference on Intelligent Robots and Systems*, Edmonton, Canada, August 2005, pp. 622– 627.

[4] L. Zhang, Y. Kim, and D. Manocha, "A hybrid approach for complete motion planning," in *Intl. Conference on Intelligent Robots and Systems*, San Diego, California, October 2007, pp. 7–14.

[5] I. A. Sucan and S. Chitta. Moveit. [Online]. Available: http://moveit.ros.org

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

[7] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "Obprm: An obstacle-based prm for 3d workspaces," in *Intl. Workshop on the Algorithmic Foundations of Robotics*, 1998.

[8] C. Nissoux, T. Simeon, and J.-P. Laumond, "Visibility based probabilistic roadmaps," in *Intl. Conference on Intelligent Robots and Systems*, vol. 3, Kyongju, Korea, October 1999, pp. 1316–1321.

[9] D. Nieuwenhuisen and M. H. Overmars, "Useful cycles in probabilistic roadmap graphs," in *IEEE Intl. Conference on Robotics and Automation*, New Orleans, Louisiana, April 2004.

[10] E. Plaku, M. Y. Vardi, and L. E. Kavraki, "Discrete search leading continuous exploration for kinodynamic motion planning," in *Robotics: Science and Systems*, W. Burgard, O. Brock, and C. Stachniss, Eds. Atlanta, Georgia: MIT Press, June 2007, pp. 326–333.

[11] C. Suh, T. T. Um, B. Kim, H. Noh, M. Kim, and F. C. Park, "Tangent space RRT: A randomized planning algorithm on constraint manifolds," in *IEEE Intl. Conference on Robotics and Automation*, Shanghai, May 2011, pp. 4968–4973.

[12] D. Berenson, S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *IEEE Intl. Conference on Robotics and Automation*, Kobe, Japan, May 2009, pp. 625–632.

[13] S. Dalibard and J.-P. Laumond, "Control of probabilistic diffusion in motion planning," in *Algorithmic Foundations of Robotics VIII*. Springer, STAR 57, 2009, pp. 467–481.

[14] B. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

[15] S. Niekum, S. Chitta, A. Barto, B. Marthi, and S. Osentoski, "Incremental semantically grounded learning from demonstration," in *RSS*, 2013.

[16] T. S. Cambon, J.-P. Laumond, J. Corts, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *International Journal of Robotics Research*, vol. 23, no. 7, pp. 729–746, 2004.

[17] F. Gravot, S. Cambon, and R. Alami, "asymov: A planner that deals with intricate symbolic and geometric problems," in *Intl. Symposium on Robotics Research*, 2003, pp. 100–110.

[18] J. Guitton and J.-L. Farges, "Taking into account geometric constraints for task-oriented motion planning," in *ICAPS Workshop on Bridging the gap Between Task And Motion Planning (BTAMP)*, 2009.

[19] K. Hauser and J.-C. Latombe, "Integrating task and prm motion planning: Dealing with many infeasible motion planning queries," in *ICAPS Workshop on Bridging the Gap Between Task and Motion Planning*, 2009.

[20] I. A. Şucan and L. E. Kavraki, "Mobile manipulation: Encoding motion planning options using task motion multigraphs," in *IEEE Intl. Conference on Robotics and Automation*, Shanghai, May 2011, pp. 5492–5498.

[21] R. Bohlin and L. Kavraki, "Path planning using Lazy PRM," in *IEEE Intl. Conference on Robotics and Automation*, San Francisco, April 2000, pp. 521–528.

[22] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics and Automation Magazine*, 2012. [Online]. Available: http://ompl.kavrakilab.org

[23] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE Intl. Conference on Robotics and Automation*, San Francisco, April 2000, pp. 995–1001.