APPEARED IN
Proc. IEEE International Symposium on Assembly and Task Planning, pp. 280–285, 1999

Placing a Robot Manipulator Amid Obstacles for Optimized Execution

David Hsu Jean-Claude Latombe Stephen Sorkin
Computer Science Department, Stanford University
Stanford, CA 94305, U.S.A.
{*dyhsu, latombe, ssorkin*}*@cs.stanford.edu*

Abstract

This paper presents an efficient algorithm for optimizing the base location of a robot manipulator in an environment cluttered with obstacles, in order to execute specified tasks as fast as possible. The algorithm uses randomized motion planning techniques and exploits geometric "coherence" in configuration space to achieve fast computation. The performance of the algorithm is demonstrated on both synthetic examples and real-life CAD data from the automotive industry. The computation time ranges from under a minute for simple problems to a few minutes for more complex ones.

1 Introduction

The base placement of a robot manipulator is an important issue in many robotics applications. Given a description of a robot manipulator and its environment, the goal is to find a base location for the manipulator so that specified tasks are executed as efficiently as possible. In this paper, we present an algorithm that makes use of randomized motion planning techniques to compute simultaneously a base location and a corresponding collision-free path that are optimized with respect to the execution time of tasks.

Our work is motivated mainly by robotics applications in the manufacturing industry, where the base placement of a manipulator has a big impact on the cycle time of tasks executed, for example, spot welding. An automated means to determine the best placement can both increase the throughput of workcells and reduce set-up time. Our algorithm can also be used to positioning mobile manipulators, manipulators mounted on mobile bases. For many mobile manipulators, the base remains stationary while the mounted manipulator is in motion, because of operational constraints and increased control complexity [17]. Thus positioning the mobile base for efficient operation of such a system is the same problem as the one for a fixed-base manipulator.

A minimum requirement for the base location of a manipulator is that the reachable workspace of the manipulator covers all the task points. Furthermore the robot should be placed to enable efficient task execution. Previous work on this problem usually considers the first and sometimes also the second criterion, but few systems take into account obstacles in the environment at the same time, partly because planning a collision-free path for a robot with more than four or five degrees of freedom (DOF) is a difficult problem. However, robot manipulators share the space with part feeders and various other devices in a workcell, and must avoid collision with them while in motion (see Figure 4, Scene 4). We therefore believe that it is essential to consider the impact of obstacles on the placement of robot manipulators.

Our algorithm first computes a collision-free path for an initial base location, using the randomized path planner described in [11]. This planner was chosen because it is one of the few that can efficiently plan a path for a many-DOF robot in a complex environment. Unlike most other randomized path planners, it does not pre-compute a roadmap of the whole configuration space $\mathcal{C}$, but samples only parts of $\mathcal{C}$ that are relevant to the current task. The path computed by the planner is then deformed to obtain a locally optimal path. Finally we iteratively move the base to better locations. At each iteration, we perturb the base location and recompute a new collision-free path. If the path planner were invoked every time, the computation would be prohibitively expensive. However, locally optimal paths for two different base locations are usually "close" to each other in the configuration space of the manipulator base and joints, if the two base locations are close. Exploiting this spatial "coherence", we use the path found in the previous iteration as a starting point for finding a new path in the current iteration. This allows us to save considerable computation time.

The rest of the paper is organized as follows. Section 2 briefly surveys related work. Section 3 gives a mathematical characterization of the problem. Section 4 and 5 describe the algorithm and experiments, respectively. Section 6 summarizes the results and points out direction for future research.

2 Related work

Planning a path for a robot amid obstacles is a classic problem in robotics [15]. The best complete planner [4], that is, a planner that finds a path whenever one exists and reports that none exists otherwise, takes time exponential in the number of degrees of freedom of a robot. In the last decade, several practical planners have been developed for many-DOF robots [2, 5, 9, 13]. A particularly interesting development has been the introduction of random sampling into path planning [1, 10, 11, 13]. Randomized path planners are capable of solving path planning problems for many-DOF robots in very complex environments. Some of them satisfy a property called probabilistic completeness: the probability that a path planner fails to find a path when one exist decreases exponentially with the running time.

A problem closely related to path-planning is that of finding the shortest path in an environment with obstacles. Research in computational geometry has yielded many efficient algorithms in 2-D environments for various metrics [8, pp. 445–466]. The shortest path problem in three or higher dimensions is considerably harder: it has been proven to be NP-hard in 3-D [3]. Therefore if we want to find a minimum-time path for a many-DOF manipulator, we may have to resort to approximation techniques. One possibility is to plan a collision-free path first and then deform the path iteratively to reduce the execution time [18]. This is the approach taken here. Alternatively, for a simple robot with two or three DOFs, one may discretize the state space of the robot and search this space for an approximation to the minimum-time path [6].

Several variants of the robot placement problem have been proposed in the literature. Seraji considers the placement of a 7-DOF Robotics Research arm by analyzing its reachability [17]. Kolarov presents an algorithm that can place a robot made up of n telescoping links amid obstacles in a planar environment [14]. Ozedou formulates the base placement problem as that of kinematic synthesis and solves it with general optimization techniques [16]. These work addresses the placement problem mainly from the reachability point of view. Feddema proposes an algorithm that places a manipulator for minimum-time joint-coordinated motion [7], but he assumes an environment with no obstacles. Hwang and Watterberg's formulation of the placement problem [12] is more closely related to ours. Their algorithm discretizes the space of all base locations and uses a path planner to exhaustively search the space in order to find the optimal solution. They report that the exhaustive search took 50 hours on a grid of 175 base locations for an environment with relatively simple geometry.

3 Preliminaries

Suppose that the base of a manipulator $\mathcal{A}$ is fixed. A *configuration* of $\mathcal{A}$ is a set of parameters that completely determines the position and orientation of all rigid parts of $\mathcal{A}$. Typically these parameters are the joint angles of $\mathcal{A}$. A configuration q is *free* if $\mathcal{A}$ does not overlap with obstacles when placed at q. The set of all configurations forms the configurations space $\mathcal{C}$, and the set of all free configurations forms the free space $\mathcal{F}$, a subset of $\mathcal{C}$. The complement of $\mathcal{F}$, $\mathcal{C}\backslash\mathcal{F}$, is called the configuration-space obstacle (*C-obstacle*).

A path in $\mathcal{C}$ is a continuous mapping from $[0, 1]$ into $\mathcal{C}$. The cost of a path can be measured in many ways, the most important ones being the time and energy that it takes a manipulator to execute the path.

If the end-effector frame (position and orientation) of a manipulator $\mathcal{A}$ is T, then for a given base location, we can solve for a configuration of the manipulator joints that achieves T via inverse kinematics (IK). The solutions define a mapping $q\colon X \to \mathcal{C}$ from the space X of

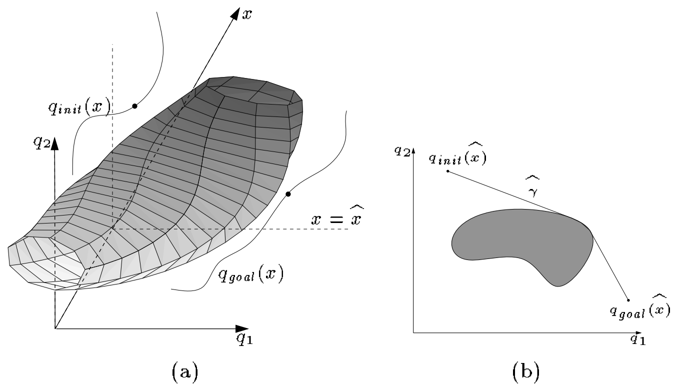

Figure 1. The base placement problem in the configuration space. The robot is a two-link planar arm with joint angles q_1 and q_2. The base of the robot lies on a line parameterized by x. (a) Schematic drawing of the configuration space of the robot base and joints. Shaded parts indicate obstacles. (b) Cross-section for the optimal base location $\widehat{x}$ and the optimal path $\widehat{\gamma}$.

all base locations to the space $\mathcal{C}$ of manipulator joint configurations. Since IK solution is not unique in general, the mapping can be one-to-many. However, for the simplicity of presentation, we will assume that the IK solution is unique, when it exists. Our algorithm easily generalizes to deal with multiple IK solutions.

In a base placement problem, we are given an initial end-effector frame T_{init} and a goal end-effector frame T_{goal}. Let $q_{init}(x)$ and $q_{goal}(x)$ be the IK solutions to T_{init} and T_{goal} at base location x, respectively. The objective is to find the best base location $\widehat{x} \in X$ such that the path between the $q_{init}(\widehat{x})$ and $q_{goal}(\widehat{x})$ has the minimum cost. We can write this more compactly as

$$\min_{x\in X} \min_{\gamma\in\Gamma_x} L(\gamma),$$

where $L(\gamma)$ is the cost of path γ, and Γ_x denotes the set of all collision-free paths such that for each $\gamma \in \Gamma_x$, $\gamma(0) = q_{init}(x)$ and $\gamma(1) = q_{goal}(x)$. Figure 1 illustrates the statement for a planar robot with two links. The base of the robot is assumed to be constrained on a line. So the configuration space of the robot base and joints is three-dimensional. For each base location, there is a cross-section of joint angles. Finding the best base placement $\widehat{x}$ is equivalent to finding the cross-section that contains the optimal path.

4 The base placement Algorithm

Our algorithm for robot placement makes use of two sub-algorithms. The first is a randomized path planner. Assume that the base location x is known. The planner generates a collision-free path γ between two configurations $q_{init}(x)$ and $q_{goal}(x)$. Usually γ contains too many unnecessary turns because of the random steps taken by the planner, and must be optimized. The second sub-algorithm takes γ as input and computes a locally optimal piecewise-linear path. The restriction to piecewise paths is not severe, as any reasonable path can be well-approximated by a piecewise-linear one. Using these two building blocks, the robot placement algorithm iteratively searches for the best base location, as described in Subsection 4.1.

Our algorithms do not represent C-obstacles explicitly. Instead, a collision-checker determines whether a configuration is free or not. To check whether a path is collision-free, we can discretize it into a sequence of configurations and regard the path free if all these configurations are free. Problems may arise if the discretization is not fine enough, but they can be eliminated by making use of the distance information returned by the collision-checker [11].

In the next three subsections, we will first look at the overall algorithm and then the two sub-algorithms in detail.

4.1 Searching for the optimal placement

In principle, given a path-planning and a path-optimization algorithm, we can search for a good base location in a brute-force way. At each candidate base location, simply call the path planner to get a collision-free path and then optimize it. However, this would be very expensive computationally due to the repeated invocation of the path planner.

Notice that if $\mathcal{B}_x$ is the C-obstacle for a manipulator placed at x, for sufficiently small Δx, $\mathcal{B}_{x+\Delta x}$ can be obtained from $\mathcal{B}_x$ by a small deformation. Therefore a collision-free path for a manipulator placed at x is likely to be collision-free in $\mathcal{C}_{x+\Delta x}$, the configuration space for the manipulator placed at $x+\Delta x$. If not, we may hope to transform it into a collision-free one by a small deformation.

Using this observation and a fast path optimization routine, we can quickly recompute an optimal path at each new base location, and iteratively move the base towards the best location. The algorithm is described in the pseudo-code below. It selects new base locations by randomly sampling the neighborhood of the best one found so far.

Algorithm *Robot Placement*

```
1.  Find a collision-free path for an initial base location $x_0$, and optimize it to obtain a piecewise-linear
    path $\gamma = (v_1, v_2, \ldots, v_N)$, where $v_1 = q_{init}(x_0)$,
    $v_2, \ldots, v_{N-1}, v_N = q_{goal}(x_0)$ are the vertices of $\gamma$.
2.  if a collision-free path $\gamma$ is found
3.    then $x \leftarrow x_0$
4.    else return FAILURE.
5.  repeat
6.     $x' \leftarrow x + \Delta x$ for some random $\Delta x$.
7.     Compute $q_{init}(x')$ and $q_{goal}(x')$ for the base location $x'$ using the manipulator IK. If $q_{init}(x')$
       or $q_{goal}(x')$ is in collision, then continue to the next iteration (line 6).
8.     if the path $(q_{init}(x'), v_2, \ldots, v_{N-1}, q_{goal}(x'))$ is collision-free
9.       then $\gamma' \leftarrow (q_{init}(x'), v_2, \ldots, v_{N-1}, q_{goal}(x'))$
10.      else sample $S$ new configurations in the neighborhoods of $v_2, v_3, \ldots, v_{N-1}$, for
              some constant $S$.
11.           Try to find a path $\gamma'$ between $q_{init}(x')$ and $q_{goal}(x')$ through the new configurations.
              Continue to the next iteration (line 6) if no collision-free path is found.
12.    Optimize $\gamma'$.
13.    if $L(\gamma') < L(\gamma)$ then $x \leftarrow x'$, $\gamma = \gamma'$.
14. until the termination condition is satisfied.
15. return $x$ and $\gamma$.
```

In lines 10-11, we find a collision-free path at the new base location by randomly sampling in the neighborhood of the path at the previous base location. This deformation technique is simple to implement and usually quite effective. However, at certain critical locations, no local deformation is sufficient to obtain a new path. In this case, we call the planner to re-plan a new path, but re-planning does not happen very frequently in practice.

The termination condition in line 14 can be implemented in various ways. A simple one is to keep track of the improvement between successive iterations and terminate if the improvement falls below some threshold. We also bound the maximum number of iterations.

Note that the outcome of the algorithm may depend on the initial location x_0 and the initial path computed by the randomized path planner. To remove this bias, one may run the algorithm several time with different values for x_0 and the initial path, and keep the best result.

4.2 Randomized path planning

The goal here is to find a collision-free path between two configurations q_{init} and q_{goal}. Our planner proceeds as follows. First sample at random in the neighborhood of q_{init}, but retain only those configurations that are connected to q_{init} by a straight-line path. Then continue sampling in the neighborhoods of configurations that have been retained. The same procedure is repeated for q_{goal}. Periodically we check for a straight-line path between a configuration path-connected to q_{init} and a configuration path-connect to q_{goal}. If there is, a path between q_{init} and q_{goal} is found. This simple algorithm is capable of solving complex path planning problems in cluttered environments in a few minutes. More details on this algorithm and implementation issues can be found in [11]. Note that if a manipulator has multiple IK solutions, we can easily extend this planner by considering multiple q_{init} and q_{goal}. Under reasonable geometric conditions on the configuration space, this planner is probabilistically complete.

4.3 Optimizing a path

The specific choice of a path optimization algorithm depends on the robot and the cost measure used. To fit into our overall algorithm, path optimization must be very fast because it is done at every candidate base location. A straightforward method for path optimization is to start with some initial path γ, sample a large number of free configurations in the neighborhood of γ, find the best path going through these configurations, and searche for a local minimum iteratively. This method is very general and can handle many different cost functions, but it can be quite slow.

For the purpose of this paper, we define the cost of a path to be the time that it takes an n-joint manipu-

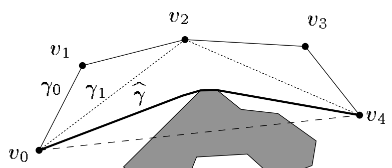

Figure 2. *Shortcut* gets stuck at γ_1, which is far from the minimum $\widehat{\gamma}$. The path γ_0 is the the input to *Shortcut*, and γ_1 shows the path after one iteration.

lator to execute the path, assuming that all joints can achieve maximum velocity in negligible amount of time. Thus if each joint of $\mathcal{A}$ has maximum speed σ_i, for $i = 1, 2, \ldots, n$, the time that it takes $\mathcal{A}$ to travel along a straight-line path in $\mathcal{C}$ between two configurations $p = (p_1, p_2, \ldots, p_n)$ and $q = (q_1, q_2, \ldots, q_n)$ is then the maximum of time required by each joint

$$d(p, q) = \max_{1 \le i \le n} \frac{|p_i - q_i|}{\sigma_i}. \qquad (1)$$

The cost of a piecewise-linear path $\gamma = (v_1, v_2, \ldots, v_N)$ is then is the sum of the cost of straight-line paths between successive vertices on γ, $L(\gamma) = \sum_{i=1}^{N-1} d(v_i, v_{i+1})$.

The function d is a scaled l_∞ metric on $\mathcal{C}$, which implies that $d(p, q) \le d(p, r) + d(r, q)$ for all $p, q, r \in \mathcal{C}$. It follows that if we replace a portion of a path by a straight line-segment in $\mathcal{C}$, the cost of the path can only decrease (or stay the same).

Lemma 1 *If p and q are two points on a path γ, and γ' is a new path obtained by replacing the part of γ between p and q by the straight-line path between p and q, then $L(\gamma') \le L(\gamma)$.*

Lemma 1 helps to characterize paths of minimum cost: there is a minimum-cost path that is locally "straight" at each point where the path is not touching the C-obstacle. More precisely, let $\widehat{\gamma}$ be a collision-free path of minimum-cost, and let Γ be the set of collision-free paths such that for each $\gamma \in \Gamma$, γ has zero curvature at each point p on γ where γ is not tangent to C-obstacle. If $\widehat{\gamma}$ is not in Γ, we can repeatedly replacing sub-paths of $\widehat{\gamma}$ by straight-line paths and in the limit reach a path $\widehat{\gamma}' \in \Gamma$. By Lemma 1, the cost of $\widehat{\gamma}'$ cannot be higher than that of γ. Hence the following lemma.

Lemma 2 *The set Γ contains a collision-free path of minimum cost.*

A simple way to optimize γ is then to recursively break γ into two sub-paths $\gamma_1 = (v_1, v_2, \ldots, v_{\lfloor N/2 \rfloor})$ and $\gamma_2 = (v_{\lfloor N/2 \rfloor}, v_{\lfloor N/2 \rfloor + 1}, \ldots, v_N)$, and check whether γ_1 and γ_2 can be replaced by straight-line paths. If they can, Lemma 1 guarantees that the cost of the new path is lower than that of γ. We will call this simple recursive procedure *Shortcut*.

Shortcut is fairly efficient, taking $O(N)$ time to execute, but unfortunately it may stop far short of reaching the minimum-cost path. In the example shown in Figure 2, it terminates when it obtains the path γ_1. No further improvement is possible because the straight-line path (v_0, v_4) is in collision. Of course, if there were more vertices on γ_1, we might be able to reach the true

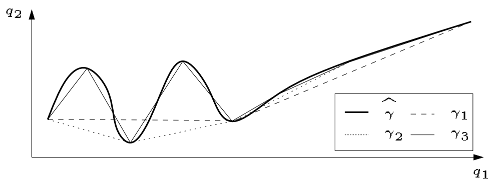

Figure 3. Piecewise linear approximations to a smooth path. The paths $\gamma_1, \gamma_2, \gamma_3$ are three-, five-, and nine-vertex piecewise-linear paths approximating $\widehat{\gamma}$. In the right portion of $\widehat{\gamma}$, all approximations are reasonable good, while in the left portion of $\widehat{\gamma}$, only γ_3 approximates $\widehat{\gamma}$ well.

minimum-cost path, but it is difficult to know in advance how many vertices are needed.

To address this problem, after *Shortcut* stops, we use an oracle to scan through each vertex v_i in the path and add additional vertices around v_i if necessary. Specifically the oracle puts two additional vertices v_l and v_r on the straight line-segments (v_{i-1}, v_i) and (v_i, v_{i+1}) respectively and try to replace the sub-path (v_l, v_i, v_r) by the straight line-segment (v_l, v_r). A bisection method is used to determine v_l and v_r so that (v_l, v_r) is collision-free. First set v_l to be the midpoint of (v_{i-1}, v_i), and v_r to be the midpoint of (v_i, v_{i+1}). If the line-segment (v_l, v_r) is not collision-free, bisect again and set v_l to be the midpoint between v_i and the previous v_l, and v_r to be the midpoint between v_i and the previous v_r. Continue until (v_l, v_r) lies completely in the free space. In general, $v_l = 1/2^k v_{i-1} + (2^k - 1)/2^k v_i$ and $v_r = 1/2^k v_{i+1} + (2^k - 1)/2^k v_i$, at kth step for $k = 1, 2, \ldots$ The procedure is guaranteed to terminate because v_i is a free configuration and hence there exists an open ball B that contains v_i and lies entirely in the free space. Once both v_l and v_r are inside B, the line-segment v_l and v_r must be collision-free, since B is convex.

After the oracle adds additional vertices to the path, *Shortcut* is invoked again. The process terminates when no further improvement is possible. *Shortcut* with an oracle will be referred to as *Adaptive Shortcut*.

We can shed some light on the efficiency of *Adaptive Shortcut* by analyzing the space of paths that it operates on. Let F_i be the space of piecewise linear paths having i vertices. Any path with i vertices can also be represented by a path with j vertices for $j \ge i$. Hence $F_i \subset F_j$ for $i \le j$; $F_i, i = 2, 3, \ldots$ form a nested sequence of function spaces. Let $F = \cup_{i=2}^{\infty} F_i$ be the space of all piecewise-linear paths. We are interested in $\widehat{\gamma}$, the optimal path in F. If we restrict the space of paths being considered to F_i for some fixed i, $\widehat{\gamma}_i$, the optimal path in F_i, remains a good approximation to $\widehat{\gamma}$ provided i is sufficiently large. However, a large i means more vertices (variables) needed to represent a path, and thus the optimization procedure may take longer to converge to a minimum. On the other hand, if i is too small, $\widehat{\gamma}_i$ can be a poor approximation to $\widehat{\gamma}$. Furthermore, different portions of $\widehat{\gamma}$ may have different levels of smoothness. On the part where $\widehat{\gamma}$ is smooth, a few vertices are enough to approximate it well; on the part where $\widehat{\gamma}$ varies widely, many more vertices are needed. See Figure 3 for an example in two dimensions. In *Adaptive Shortcut*, the *Shortcut* procedure removes unnecessary vertices when replacing sub-paths by straight line-segments, and the oracle adds more vertices where needed. By moving up and down among various spaces $F_2, F_3, \ldots$, *Adaptive Shortcut* quickly converges to a good approximation to $\widehat{\gamma}$. This algorithm is inspired by multi-resolution representation of curves in computer graphics [19].

In practice, the cost of a path that we have defined does not take into account manipulator dynamics and is only an approximation to the time that it takes a manipulator to execute a path. It is a good approximation if the manipulator can reach the maximum speed quickly. If dynamics must be considered, we will have to resort to the general technique mentioned at the beginning of the subsection or some other path optimization methods, but they are likely to be much slower than *Adaptive Shortcut*.

5 Computed examples

We have implemented our algorithm in C++ and tested it on several data sets. Four examples in our test suite are shown here (Figure 4). They vary in the complexity of workspace and motion needed to complete the specified tasks. Scene 1 is a simple blocks-world, and the motion required of the robot is straightforward. Scene 2 is much more complex in terms of both the workspace geometry, and the motion required. The robot has to go through openings in the window and sun-roof of the car in order to reach the task-points. Scene 3 has relatively simple geometry, but in one of the tasks (p2), the robot has to execute complicated maneuvers in order to pull the big end-effector through a small rectangular hole. Scene 4 is a large CAD model containing about 72,100 triangles. The robot has to maneuver among fixturing devices and reach under the side-panel of a car in order to access the task-points. Scene 1-3 were synthesized for testing; Scene 4 was derived from CAD data provided to us by General Motors Corporation. For each of these scenes, we specify a number of tasks in the form of a pair of initial and goal end-effector frames (Cartesian frames at the wrist-point of the robot), labeled by p1, p2, or p3 in the figure.

Two types of robot are used in the tests. Scene 1 and Scene 2 use a PUMA robot that has six DOFs and about 460 triangles describing its geometry. Scene 3 and Scene 4 use a modified version of a FANUC-200 robot. We have replaced the four-bar linkage of the FANUC-200 by a simple revolute joint in order to simplify the IK solution. The geometry of the robot is mostly preserved. The modified robot has six DOFs and is modeled by $1,260$ triangles.

The results of our experiments are shown in Table 1. The running time was collected on an SGI Indigo 2 workstation with an 195 MHz R10000 processor and 384 MB memory. In all the tests, our algorithm took only a few minutes to finish the computation, and less than a minute for simpler problems. The cost of the paths was reduced by about 50% by choosing a good base location and optimizing the path. Column 2 of the table lists the total number of triangles contained in all the objects in the environment, including the

Table 1. Running time and cost of paths for tested scenes.

Scene	N_{tri}	Task	T_{plan}	N_{plan}	T_{opt}	N_{opt}	C_{init}	C_{i-opt}	C_{final}
			(sec.)		(sec.)				
1	560	p1	0.3	309	15.1	9955	1.36	0.52	0.32
		p2	0.3	362	8.8	7016	1.08	0.48	0.39
		p3	0.3	353	9.0	6831	1.26	0.77	0.50
2	21400	p1	3.4	2831	39.8	17647	2.25	1.51	1.16
		p2	4.2	3806	31.7	10053	2.57	1.53	1.08
		p3	16.5	14794	34.1	10255	2.80	1.69	1.44
3	1368	p1	4.4	4331	25.5	17859	1.13	0.84	0.48
		p2	197.0	210829	55.4	23566	2.17	1.43	1.30
4	72100	p1	11.4	7425	107.4	24436	1.78	1.18	0.82
		p2	25.1	28263	133.6	19453	1.24	0.79	0.59

robot. It gives an idea of the complexity of the environment. Columns 4 and 5 give the running time and number of collision checks needed to find an initial collision-free path. The randomized motion planning algorithms performed very well in these examples. All problems except for one were solved in less than 30 seconds. In the case where the planner took a relatively long time, the robot must pull the welding-gun (end-effector) through a small hole, a very challeenging situation for a randomized motion planner. Columns 6 and 7 show the running time and number of collision checks spent searching for the best base placement and a corresponding collision-free path. The running time ranges from a few seconds to 2-3 minutes. Recall that at each new base location, the algorithm must find a new collision-free path and optimize it. Both operations require a large number of collision checks and are very expensive. The computation time was siginificantly reduced by exploiting the spatial coherence in the configuration space as we have discussed previously. The last three columns of the table list respectively the cost of the initial path, the cost of the optimized path at the initial base location, and the cost of the path at the best base location found. By comparing the costs in columns 8 and 9, we see that *Adaptive Shortcut* was able to reduce the cost of paths by about 25-60%. Finding a good base location further reduced the cost by additional 10-30%.

6 Conclusion and future work

This paper presents an algorithm that computes a locally optimal base location and a corresponding collision-free path for a robot manipulator to move between two end-effector frames in minimum amount of time. We have tested this algorithm on both synthesized examples and real-life CAD data from the automotive industry. Experiments show that our algorithm can significantly reduce the cycle time by choosing a good base location and optimizing the path, and therefore improve the productivity of a workcell. In all our experiments, the computation was completed in a few minutes. The efficiency of our algorithm, we believe, is largely due to the randomized motion planning techniques used. These techniques have allowed us to give a more realistic formulation of the robot placement problem that takes into account obstacles in the environment.

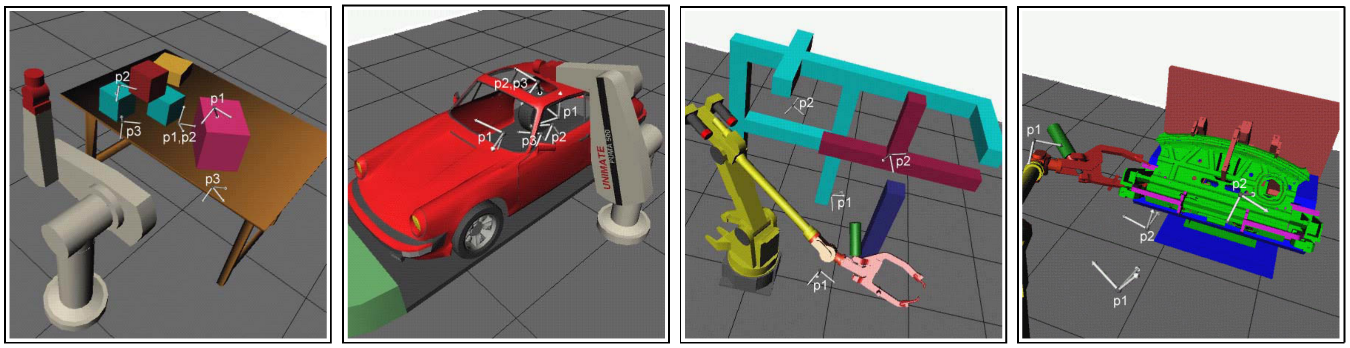

Figure 4. Four test scenes. (Larger images are available at `http://robotics.stanford.edu/people/dyhsu/projects/placement`)

Currently our algorithm considers a minimum-cost path between two end-effector frames. An immediate extension of the problem is to consider n end-effector frames. Given a set of end-effector frames and a partial order on them so that some frames have to be visited before others, we would like to find a path that visits each frame exactly once and obeys the partial order. This is a variant of the traveling salesman problem, which has been studied extensively, and many heuristic and approximation algorithms are available.

Although our algorithm was originally developed for the base placement problem, the approach can potentially be applied to kinematic synthesis problems, where one needs to find a small set of parameters to configure a robot. Application of our algorithm to this more general class of problems is currently under investigation.

Acknowledgments This research has been partially supported by ARO MURI grant DAAH-04-96-1-007 and by a grant from the Stanford Intergrated Manufacturing Association (SIMA). David Hsu is a recipient of the Microsoft Graduate Fellowship. The authors thank Rajeev Motwani for his help in several aspects of this work. They also thank Charles Wampler at General Motors Corp. for providing CAD models and Deneb Robotics Inc. for making the IGRIP software available to us.

References

[1] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Proc. Workshop on Algorithmic Foundation of Robotics*, 1998.

[2] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. of Robotics Research*, 10(6):628–649, 1991.

[3] J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *Proc. IEEE Symp. on Foundations of Computer Science*, pp. 49–60, 1987.

[4] J. Canny, J. Reif, B. Donald, and P. Xavier. On the complexity of kinodynamic planning. In *Proc. IEEE Conf. on Foundations of Computer Science*, pp. 306–316, 1988.

[5] P. C. Chen and Y. K. Hwang. SANDROS: A motion planner with performance proportional to task difficulty. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 2346–2353, 1992.

[6] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *J. ACM*, 40(5):1048–1066, 1993.

[7] J. T. Feddema. Kinematically optimal robot placement for minimum time coordinated motion. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 3395–3400, 1996.

[8] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press, New York, 1997.

[9] K. Gupta and X. Zhu. Practical motion planning for many degrees of freedom: a novel approach within sequential framework. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 2038–2043, 1994.

[10] T. Horsch, F. Schwarz, and H. Tolle. Motion planning for many degrees of freedom - random reflections at C-Space obstacles. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 3318–3323, 1994.

[11] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 2719–2726, 1997.

[12] Y. K. Hwang and P. A. Watterberg. Optimizing robot placement for visit-point tasks. In *Proc. AAAI Workshop on Artificial Intelligence for Manufacturing*, pp. 81–86, 1996.

[13] L. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration space. *IEEE Trans. on Robotics and Automation*, 12(4):566–580, 1996.

[14] K. Kolarov. Algorithms for optimal design of robots in complex environments. In K. Goldberg et al., editors, *Algorithmic Foundations of Robotics*, pp. 347–369. A. K. Peters, 1995.

[15] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[16] F. B. Ouezdou. General method for kinematic synthesis of manipulators with task specifications. *Robotica*, 15(6):653–661, 1997.

[17] H. Seraji. Reachability analysis for base placement in mobile manipulators. *J. of Robotic Systems*, 12(1):29–43, 1995.

[18] Z. Shiller and S. Dubowsky. On computing the global time-optimal motions of robotic manipulators in the presence of obstacles. *IEEE Trans. on Robotics and Automation*, 7(6):785–797, 1991.

[19] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. *Wavelets for Computer Graphics*. Morgan Kaufmann, San Francisco, CA, 1996.