Proceedings of the 2002 IEEE/RSJ
Intl. Conference on Intelligent Robots and Systems
EPFL, Lausanne, Switzerland • October 2002

Probabilistic Motion Planning for Redundant Robots along Given End-Effector Paths

Giuseppe Oriolo Mauro Ottavi Marilena Vendittelli

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Eudossiana 18, 00184 Roma, Italy
{oriolo,venditt}@dis.uniroma1.it, mauroottavi@libero.it

Abstract

We consider the problem of planning collision-free motions for a redundant robot whose end-effector must travel along a given path. Although collision avoidance is one of the main reasons for introducing kinematic redundancy in manipulators, the planning methods so far proposed for this particular problem are neither efficient nor complete. In this paper, we introduce some algorithms that may be considered as an extension of probabilistic planning techniques to the problem at hand. All the algorithms are based on the same simple mechanism for generating random samples of the configuration space that are compatible with the end-effector path constraint. Experimental results illustrate the performance of the planners.

1 Introduction

With respect to conventional robots, redundant robots exhibit an increased dexterity that allows to pursue additional objectives [1], among which the most important is obstacle avoidance. In many applications, the end-effector must trace a given path to complete the task specified by a higher-level module (e.g., for laser cutting, spray painting or inspection). A lower-level planner is then in charge of generating joint motions that realize the end-effector path while guaranteeing that the manipulator links avoid collision with workspace obstacles or between themselves (self-collision). We refer to this problem as Motion Planning along End-effector Paths (MPEP).

Most researchers have attacked the MPEP problem by formulating it as a special case of redundancy resolution and using either kinematic control schemes [2] or optimal control techniques [3]. Both these approaches work at the velocity level, first using a closed-form for the inverse differential kinematics parameterized by the available degrees of freedom, and then choosing the parameters in such a way that some configuration-dependent criterion is improved (in kinematic control) or optimized (in optimal control). For collision avoidance, obstacle distance functions are commonly used as criteria. However, none of the above solutions to the MPEP problem can be considered to be satisfactory. In fact, kinematic control schemes are based on local optimization techniques (such as the projected gradient method) and may fail to solve even simple problems if local constrained minima appear; while an optimal control formulation of MPEP leads to a nonlinear TPBVP whose solution can only be seeked (without guarantee of success) via numerical techniques.

On the other hand, the motion planning literature of recent years has been dominated by probabilistic algorithms, that can efficiently plan point-to-point motions in high-dimensional configuration spaces guaranteeing a property called probabilistic completeness (i.e., the probability of finding a solution to a solvable problem approaches one as the planning time increases). Many techniques belonging to this family have been proposed, ranging from multiple-query planners which build a connectivity roadmap of the free configuration space [4] to single-query planners that aim at representing only the portion of this space useful for a specific instance of the problem [5, 6]. While the basic tool of all these methods is the random sampling of the free configuration space, they differ in the exploration strategy as well as in the use of various heuristics to guide the search.

The objective of this paper is to show that the same ideas can be exploited to devise planning algorithms for the MPEP problem. Probabilistic planners for robots with closed kinematic chains have been proposed in [7, 8]; although the problem therein considered is related to MPEP, the developed methods do not directly apply to its solution. The pose-to-pose planning problem (without end-effector path constraint) for redundant robots was considered in [9].

The paper is organized as follows. In the next section, we give a precise formulation of the MPEP

problem for redundant robots and clarify what we consider to be a solution. Then, the proposed planners are described, isolating first the basic tools common to all of them. Experimental results for problems of increasing complexity are finally presented to illustrate the performance of the algorithms.

2 Problem Formulation

Consider a fixed-base redundant manipulator with n joints whose end-effector task is specified by m variables (position and possibly orientation). The direct kinematics is expressed as

$$p = f(q), \tag{1}$$

where $p \in \mathbb{R}^m$ is the end-effector *pose* and $q \in \mathbb{R}^n$ is the joint vector[1]. A desired end-effector path $p(\sigma)$ is assigned, with $\sigma \in [0,1]$ the path parameter. For the problem to be well-posed, we assume that:

$$p(\sigma) \in \mathcal{T}, \quad \forall \sigma \in [0,1],$$

where $\mathcal{T} \subset \mathbb{R}^m$ is the *dextrous task space*, defined as the set of end-effector poses that can be realized by ∞^{n-m} joint configurations[2]. The *dextrous workspace* W, i.e., the positional part of $\mathcal{T}$ (a subset of $\mathbb{R}^2$ or $\mathbb{R}^3$ depending on whether we are considering planar or spatial motions) is populated by obstacles.

The MPEP problem is to find a joint path $q(\sigma)$ satisfying

$$p(\sigma) = f(q(\sigma)), \quad \forall \sigma \in [0,1],$$

and such that no collision occurs between the robot linkage and the obstacles along the path.

Note the following points:

- Even if the end-effector path is by assumption contained in $\mathcal{T}$, a solution to the MPEP problem may exist or not depending on the particular obstacle placement, i.e., on the connectivity of the portion of the free configuration space that is compatible with the path constraint.
- In view of the above formulation, and particularly of the synchronization between the end-effector and the joint path, the robot is not allowed to perform *self-motions* (i.e., joint motions that do not move the end-effector) along the path, unless they are already provided for by the end-effector path specification by setting:

$$p(\sigma) = p(\sigma_1), \quad \forall \sigma \in [\sigma_1, \sigma_2].$$

- Depending on the application, an initial joint configuration $q(0)$ such that $p(0) = f(q(0))$ may or may not be assigned. For example, the first may be the case when the task trajectory is planned on the basis of sensory information gathered at the current robot posture. On the other hand, the determination of $q(0)$ will be typically left to the planning algorithm when the end-effector task is assigned off-line. The first version of the problem is clearly more constrained (and thus easier to solve, provided that a solution exists) than the second.
- The above formulation may be immediately extended to account for the existence of joint limits and/or self-collision avoidance.

We seek a solution to the MPEP problem in the form of a sequence of collision-free joint configurations:

$$\{q(\sigma_0), q(\sigma_1), \ldots, q(\sigma_{s-1}), q(\sigma_s)\}, \quad \sigma_0 = 0, \sigma_s = 1,$$

with the σ_i's equally spaced and $p(\sigma_i) = f(q(\sigma_i))$. The integer s is called *path sampling*. A continuous joint path is derived from the sequence by generating *local paths* via interpolation. Here, we consider linear interpolation between configurations. While this may lead to violations of the end-effector path constraint between successive configurations, the entity of such violation can be reduced at will by increasing the sampling s, provided that the configurations are sufficiently 'close'. In any case, interpolation schemes that account for the path constraint may be easily designed, e.g., by using pseudoinverse control [2]. As for guaranteeing collision avoidance along local paths, this will be delegated to the collision checking algorithm (more on this in Sect. 3.1).

3 Planning Algorithms

This section describes the various algorithms we have developed for the solution of the MPEP problem. Although they are conceptually different, they make use of the same tools, i.e., two procedures which respectively perform random sampling of self-motion manifolds and collision checking. Before presenting the planning algorithms, we shall therefore discuss in some detail these procedures.

3.1 Basic tools

Under the assumptions of the previous section, each end-effector pose $p \in \mathcal{T}$ can be realized by ∞^{n-m} joint configurations, which represent the so-called *self-motion manifold*[3]. The algorithms to be proposed in the next section are all based on the same idea, i.e., sampling the self-motion manifold of each pose $p(\sigma_i)$ of the sequence in a random fashion. Our sampling mechanism is essentially the same used in [8] for guaranteeing the closure constraint.

Reorder and partition the joint vector as $q = (q^b, q^r)$, where $q^b \in \mathbb{R}^m$ are the *base* joints and $q^r \in \mathbb{R}^{n-m}$

[1] In general, both p and q are properly defined over manifolds, which are only locally diffeomorphic to euclidean spaces. In this paper, we consider euclidean spaces for simplicity of exposition, but all our developments apply to the general case.

[2] $\mathcal{T}$ does not contain its boundary, which includes the unavoidable singularities realized by a single configuration.

[3] To be precise, the inverse image of any point $p \in \mathcal{T}$ is in general a finite number of disjoint manifolds [10].

are the *redundant* joints. Assume that the value of the redundant joint variables $q_i^r = q^r(\sigma_i)$ is randomly chosen (keeping into account the possible existence of joint limits). For each value of $p_i = p(\sigma_i)$, $i = 0, \ldots, s$, there exist a finite number of base joint placements $q_i^b = q^b(\sigma_i)$ such that $p_i = f(q_i^b, q_i^r)$, computed by inverting the direct kinematic map (1) of the considered manipulator with $q^r = q_i^r$. For example, if $m = 2$ (planar robots with end-effector positioning task), we have at most 2 possible base joint placements (the so-called *elbow up/elbow down* postures) for each choice of the redundant joint values. Clearly, depending on the chosen value for q_i^r, it may happen that no value of q_i^b exists which is compatible with the end-effector pose p_i.

According to the above strategy, the procedure RAND_CONF that generates a random sample of the self-motion manifold corresponding to p_i is described in pseudocode as follows.

```
RAND_CONF(p_i, q_bias)
  q_i^r ← RAND_RED(q_bias)
  q_i^b ← INV_KIN(p_i, q_i^r, q_bias)
  if INV_KIN_FAIL
    Return RAND_CONF_FAIL
  else Return q_i ← (q_i^b, q_i^r)
```

An additional optional argument q_{bias} appears in the procedure. When q_{bias} is present, RAND_CONF returns (if successful) a configuration q_i such that $p_i = f(q_i)$ and $\|q_i - q_{\text{bias}}\|_\infty < d$, where d is the maximum allowed displacement for each joint. This corresponds to biasing with q_{bias} the uniform distribution characterizing the randomly generated samples. In particular, INV_KIN takes as input the pose p_i and the redundant joint variables q_i^r generated through a limited random perturbation of q_{bias}^r, and seeks an inverse solution q_i^b for the base joints satisfying the displacement constraint. If no such solution exists (either because no inverse solution exists for the chosen q_i^r or because all solutions violate the displacement constraint) the boolean variable INV_KIN_FAIL becomes *true*. If q_{bias} is not present, the generated sample of the self-motion manifold is not biased by any configuration. Finally, when RAND_CONF is invoked with no arguments, a completely random configuration (in general, not belonging to any self-motion manifold) is generated.

The reason for biasing the random generation of q_i with a given configuration q_{bias} is that all the algorithms to be presented work in an incremental fashion, trying to build a connectivity roadmap from the initial end-effector pose. When a sample configurations q_i on the self-motion manifold corresponding to p_i has been randomly generated, it is used as q_{bias} for the next self-motion manifold in order to guarantee that q_{i+1} will be sufficiently close to q_i. As discussed at the end of Sect. 2, this will imply that the end-effector constraint violation between q_i and q_{i+1} due to the use of linear local paths is reduced.

The second procedure used by all algorithms is NO_COLL. When invoked with a single argument q_i, it performs a collision check (possibly including self-collisions) and returns *true* if q_i is safe. When invoked with two arguments (q_i, q_j), it performs a collision check on both configurations as well as on the linear path joining them. In particular, our function makes use of an *incremental* collision checker, i.e., an algorithm that, if q_i is found to be safe, computes an ϵ such that any configuration on the line connecting q_i to q_j and within euclidean distance ϵ from q_i is certainly safe. Hence, the fact that successive configurations built by RAND_CONF are never further than $d\sqrt{n}$ implies that the linear path from q_i to q_j is certainly safe whenever $d\sqrt{n} < \epsilon$, thus optimizing the performance of the collision checker.

3.2 Greedy Planner

The core of the first algorithm is the STEP function which, given two generic poses p_i, p_k $(0 \le i < k \le s)$ belonging to the end-effector sequence and a configuration q_i on the self-motion corresponding to p_i, tries to build a (sub)sequence of configurations $\{q_i, \ldots, q_k\}$ connecting p_i to p_k and such that collisions are avoided along the path. If successful, STEP returns the sequence in the variable PATH.

```
STEP(i, p_i, q_i, k)
 for j = i to k − 1 do
  l ← 0; Succ ← 0;
  while l < MAX_SHOTS and !Succ do
    q_{j+1} ← RAND_CONF(p_{j+1}, q_j);
    if !RAND_CONF_FAIL and NO_COLL(q_j, q_{j+1})
      Succ ← 1; ADD_TO_PATH(q_{j+1});
    l ← l + 1;
  if l = MAX_SHOTS
    Return STEP_FAIL
  else
    j ← j + 1;
 Return PATH
```

MAX_SHOTS represents the upper bound to the number of calls to RAND_CONF(p_{j+1},q_j) for each end-effector pose p_j. If RAND_CONF succeeds in finding a configuration q_{j+1} realizing p_{j+1} and sufficiently close to q_j, the linear path between q_j and q_{j+1} is verified to be collision-free; in this case, q_{j+1} is added to the current sequence through the ADD_TO_PATH function. If the maximum number of trials of RAND_CONF is exceeded, the procedure returns STEP_FAIL.

A direct approach to the solution of the MPEP problem is to devise a greedy algorithm based on iterated calls to the STEP function with p_0, p_s as subsequence extrema and randomly generated q_0.

GREEDY algorithm

$j \leftarrow 0$;
while $j <$ MAX_ITER **and** STEP_FAIL **do**
  $q_0 \leftarrow$ RAND_CONF(p_0);
  STEP($0, p_0, q_0, s$);
  $j \leftarrow j + 1$;
if !STEP_FAIL
  Return PATH
else
  Return FAILURE

Given the initial pose p_0 on the end-effector path, RAND_CONF(p_0) generates an initial configuration q_0 as described in the previous section (if failure is returned, RAND_CONF(p_0) is called again until a configuration is generated). The function STEP is then invoked to search for a sequence of joint configurations guaranteeing collision-free motion while the end-effector moves from p_0 to p_s. In case of successful search, the path found by STEP is returned as a solution. If STEP fails, a new initial configuration q_0 is generated and STEP starts a new search from q_0, provided that the maximum number of iterations MAX_ITER has not been exceeded.

GREEDY implements a depth-first search, as for any initial configuration q_0 a sequence of random configurations (one for each self-motion manifold, and each biased by the previous one) is generated that is discarded if STEP does not succeed in reaching the next self-motion manifold. Experiments have shown that this planner is very effective in dealing with easy problems (see Sect. 4), essentially due to the end-effector path constraint, which greatly reduces the admissible internal motions of the robot once a q_0 has been chosen. Still, the only possible way to backtrack for this planner is to generate a new q_0, and this may prove inefficient in more complex problems.

3.3 RRT-Like planner

To overcome the limitations of the depth-first strategy used by GREEDY, one may try to generate more than one random sample for each self-motion manifold and to connect configurations on successive manifolds by local paths. This exploratory behavior can be achieved by adapting the notion of Rapidly-exploring Random Tree (RRT) to the problem at hand, as shown below. The reader is referred to [5] for a detailed description of the original concepts.

Our algorithm tries to expand a tree τ rooted at q_0, a random sample of the p_0 self-motion manifold, until the self-motion manifold of p_s is reached. If the expansion fails a certain number of times, a different q_0 is generated and another tree is built, until the maximum number of iterations is exceeded. If a tree connecting p_0 to p_s is found, a solution path is extracted by graph search techniques.

RRT_LIKE algorithm

$j \leftarrow 0$;
while $p_{\text{new}} != p_s$ **and** $j <$ MAX_ITER **do**
  $q_0 \leftarrow$ RAND_CONF(p_0);
  CREATE(τ, q_0);
  $i \leftarrow 0$;
  repeat
    $p_{\text{new}} \leftarrow$ EXTEND_LIKE(τ);
    $i \leftarrow i + 1$;
  until $p_{\text{new}} = p_s$ **or** $i =$ MAX_EXT
  $j \leftarrow j + 1$;
if $p_{\text{new}} = p_s$
  Return τ
else
  Return FAILURE

The procedure EXTEND_LIKE is defined as follows.

EXTEND_LIKE(τ)

$q_{\text{rand}} \leftarrow$ RAND_CONF;
$(q_{\text{near}}, k) \leftarrow$ NEAR_NODE(q_{rand}, τ);
$q^r_{\text{new}} \leftarrow$ NEW_CONF($q^r_{\text{near}}, q_{\text{rand}}$);
$q^b_{\text{new}} \leftarrow$ INV_KIN($p_{k+1}, q^r_{\text{new}}, q_{\text{near}}$);
$q_{\text{new}} \leftarrow (q^r_{\text{new}}, q^b_{\text{new}})$;
if !INV_KIN_FAIL **and** NO_COLL($q_{\text{near}}, q_{\text{new}}$)
  ADD_NODE(τ, q_{new});
  ADD_EDGE($\tau, q_{\text{near}}, q_{\text{new}}$);
  Return p_{k+1}
else
  Return NULL

First, RAND_CONF is called with no arguments to find a random configuration $q_{\text{rand}} = (q^b_{\text{rand}}, q^r_{\text{rand}})$, and NEAR_NODE identifies q_{near} among the nodes in τ as the one closest to q_{rand} with respect to the redundant joint variables only, returning also the index k of the end-effector pose p_k to which q_{near} was associated. Then, NEW_CONF computes q^r_{new} by displacing q^r_{near} of a stepsize $d\sqrt{n}$ along the line connecting q^r_{near} to q^r_{rand} (see Fig. 1), and INV_KIN is invoked in order to generate a q^b_{new} such that $p_{k+1} = f(q^b_{\text{new}}, q^r_{\text{new}})$ and the displacement constraint between q_{near} and q_{new} is satisfied. At this point, the linear path joining q_{near} to q_{new} can be checked for collision; if the result is negative, τ is expanded and p_{i+1} is returned.

3.4 Variations on RRT_LIKE

The expansion of the tree τ toward randomly selected directions confers to the RRT_LIKE planner a definite exploratory attitude which, for the MPEP problem, could prove to be inefficient due to the strong constraint represented by the assigned end-effector path. The performance of RRT-based planners can be considerably improved if the tree expansion is guided by a greedy heuristic, as shown in [6]. Unfortunately, the approach presented therein does not apply directly to our problem because a

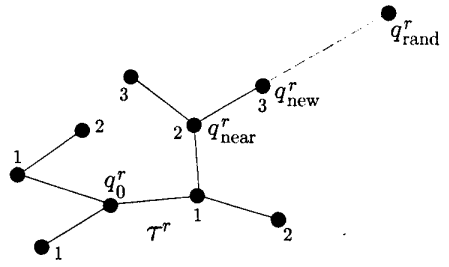

Figure 1: *The NEW_CONF procedure taking place in R^{n-m}, the redundant joint space. The integer k associated to each node identifies the end-effector pose p_k of which the node is a preimage.*

goal self-motion manifold is given here rather than a goal configuration. However, it is still possible to modify the RRT_LIKE planner by alternating depth-first searches with expansion steps. This can be done by invoking the STEP function right after the EXTEND_LIKE operation has been executed. Depending on the arguments passed to STEP, two different algorithms are obtained, called RRT_CONNECT_LIKE and RRT_GREEDY_LIKE.

In the RRT_CONNECT_LIKE planner, STEP is invoked with p_{new}, p_s as subsequence extrema. In other words, when a new configuration q_{new} is added to τ, the algorithm tries to reach the p_s self-motion manifold starting from the self-motion manifold corresponding to q_{new}. This can also be considered as a variation of the GREEDY algorithm, the difference being that in the case of STEP_FAIL the search is not resumed from a new q_0 on the self-motion manifold corresponding to p_0, but instead from the newly added configuration q_{new}. This choice gives to the planner a sort of backtracking property.

In the RRT_GREEDY_LIKE planner, the poses passed to STEP are p_l, p_{l+1}, where p_l is the closest pose to p_s reached so far by the algorithm. The algorithms tries therefore at the same time to explore the portion of configuration space consistent with the end-effector path constraint and to approach the goal self-motion manifold through a greedy search.

Indeed, it is possible to conceive another variation of RRT_LIKE in which STEP is invoked twice after EXTEND_LIKE, first as in RRT_CONNECT_LIKE and then as in RRT_GREEDY_LIKE. We call this last planner RRT_GREEDY+CONNECT.

4 Experiments

In this section, we report some planning experiments for a planar robot with six revolute joints, whose task is to move the end-effector along a given positional path ($m = 2$, $n = 6$). All the planners use the same partition of the joint vector, i.e., $q^r = (q_1, \ldots, q_4)$ and $q^b = (q_5, q_6)$. The algorithms have been implemented in C on a 900Mhz PC and integrated in the Move3D software development kit (`http://www.kineocam.com`), where collision detection is realized using the V-Collide library.

Experiments have shown that the GREEDY planner is always more effective than the others for relatively simple queries. A typical example is shown in Fig. 2, where the end-effector must follow an arc of circle. The planners' performance (averaged on 10 realizations of the planning process) is summarized in Tab. 1, including the number of collision checks during the search. Note that GREEDY requires more checks than RRT-based planners, whose performance is however hindered by the NEAR_NODE operation within EXTEND_LIKE.

The second experiment of Fig. 3 is more difficult than the first, as the robot has little space to maneuver while moving the end-effector along a circle. As shown by Tab. 2, GREEDY behaves again satisfactorily, but RRT_GREEDY achieves a better performance. The results of the pure RRT planner are quite poor, essentially because the tree expansion towards unexplored regions of the configuration space proves to be useless in view of the limited variation range for the redundant joint variables.

Finally, the results obtained for a difficult query are illustrated in Fig. 4 and Tab. 3. Here, the robot must move the end-effector inside a narrow opening between two obstacles. The performance of the GREEDY planner is in this case much worse, due to the depth-first strategy. The best results are obtained by RRT_CONNECT+GREEDY, which combines the breadth search typical of RRT_CONNECT_LIKE with a greedy heuristic.

5 Conclusions

With reference to the problem of planning collision-free motions for a redundant robot moving along a given end-effector path, we have presented some single-query probabilistic planners based on current ideas in the motion planning literature. Experimental results have shown that simple instances of the problem can be solved more efficiently if a greedy heuristic is used, whereas breadth-first search is needed to deal with more complex problems.

Among the aspects that could not be discussed here, we mention the problem of selecting the base variables for the random sampling strategy and the probabilistic completeness of the algorithm, which however can be established along the same lines of [4, 5, 8] provided that finite iteration bounds such as MAX_ITER are not enforced. Also, although our discussion was focused on fixed-base robots, it would be interesting to apply these techniques to kinematically redundant mobile robots such as nonholonomic mobile manipulators.

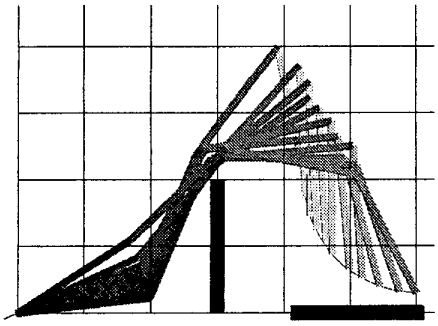

Figure 2: *First experiment*

Planner	Time(s)	# of CC
GREEDY	0.13	96.8
RRT_LIKE	0.58	52
RRT_CONNECT_LIKE	0.4	107
RRT_GREEDY_LIKE	0.53	90
RRT_GREEDY+CONNECT	0.54	110

Table 1: *First experiment: Planner performance*

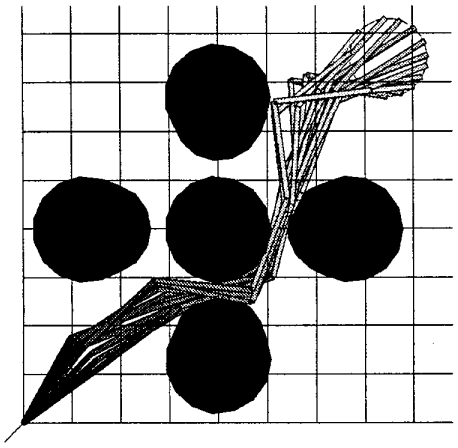

Figure 3: *Second experiment*

Planner	Time(s)	# of CC
GREEDY	11.2	13477
RRT_LIKE	75.8	53813
RRT_CONNECT_LIKE	16.6	13689
RRT_GREEDY_LIKE	10.0	10264
RRT_GREEDY+CONNECT	15.5	13426

Table 2: *Second experiment: Planner performance*

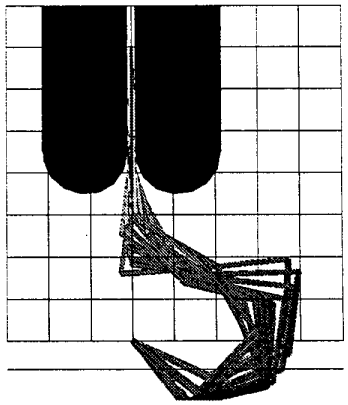

Figure 4: *Third experiment*

Planner	Time(s)	# of CC
GREEDY	107.0	296836
RRT_LIKE	45.9	101620
RRT_CONNECT_LIKE	33.5	58200
RRT_GREEDY_LIKE	22.9	55828
RRT_GREEDY+CONNECT	16.5	36058

Table 3: *Third experiment: Planner performance*

References

[1] Y. Nakamura, *Advanced Robotics: Redundancy and Optimization*, Addison-Wesley, 1991.

[2] B. Siciliano, "Kinematic control of redundant robot manipulators: A tutorial", *J. of Intelligent and Robotic Systems*, vol. 3, pp. 201–212, 1990.

[3] D. P. Martin, J. Baillieul, and J.M. Hollerbach (1989), "Resolution of kinematic redundancy using optimization techniques," *IEEE Trans. on Robotics and Automation*, vol. 5, pp. 529–533, 1989.

[4] L. Kavraki, P. Svestka, J. C. Latombe, and M. H. 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.

[5] S. M. LaValle, "Rapidly-exploring Random Trees: A new tool for path planning," Tech. Rep., Computer Science Dept., Iowa State University, 1998.

[6] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An efficient approach to single query path planning," *2000 IEEE Int. Conf. on Robotics and Automation*, pp. 995–1001, 2000.

[7] J. H. Yakey, S. M. LaValle, and E. L. Kavraki, "Randomized path planning for linkages with closed kinematic chains," *1999 IEEE Int. Conf. on Robotics and Automation*, pp. 1671–1677, 1999.

[8] L. Han and N. Amato, "A Kinematic-Based Probabilistic Roadmap Method for Closed Chain Systems," *4th Int. Work. on Algorithmic Foundations of Robotics*, pp. 233–246, 2000.

[9] J. M. Ahuactzin and K. K. Gupta, "The kinematic roadmaps: A motion planning based global approach for inverse kinematics of redundant robots," *IEEE Trans. on Robotics and Automation*, vol. 15, no. 4, pp. 653–669, 1999.

[10] J. Burdick, "On the inverse kinematics of redundant manipulators: Characterization of the self motion manifolds," *1989 IEEE Int. Conf. on Robotics and Automation*, pp. 264–270, 1989.