Motion Planning for Mobile Manipulators along Given End-effector Paths

Giuseppe Oriolo  Christian Mongillo
Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”
Via Eudossiana 18, 00184 Roma, Italy
oriolo@di.uniroma1.it, chrismongillo@infinito.it

Abstract—We consider the problem of planning collision-free motions for a mobile manipulator whose end-effector must travel along a given path. Algorithmic solutions are devised by adapting a technique developed for fixed-base redundant robots. In particular, we exploit the natural partition of generalized coordinates between the manipulator and the mobile base, whose nonholonomy is accounted for at the planning stage. The approach is based on the randomized generation of configurations that are compatible with the end-effector path constraint. The performance of the proposed algorithms is illustrated by several planning experiments.

Index Terms—Mobile manipulators, probabilistic motion planning.

I. INTRODUCTION

Mobile manipulators combine the two archetypes of robotic systems, i.e., articulated arms and mobile platforms: hence, they exhibit the dexterity and grasping capability of the former and the sensor-based mobility of the latter. Several prototypes of mobile manipulators already exist; see [1-2] for some examples. However, many research aspects still need to be addressed in order to fully exploit the potential of these mechanisms. In fact, the arm and the mobile platform are often treated as distinct entities, neglecting their kinematic and dynamic interaction, whereas their most effective use is expected to rely on the coordinated use of locomotion and manipulation functions [3].

Planning collision-free motions under task constraints is a typical problem where whole-system coordination is crucial. In many applications, the mobile manipulator is required to move the end-effector along a given path in order to realize the task specified by a higher-level module (e.g., for inspection missions with an in-hand camera, or in pick and place operations). A lower-level planner is then in charge of generating joint paths that realize the desired end-effector motion while guaranteeing that the robot avoids collisions with obstacles or with itself. We call this problem Motion Planning along End-effector Paths (MPEP).

Since mobile manipulators are kinematically redundant with respect to end-effector tasks, the MPEP problem can be attacked as an optimal redundancy resolution problem, with the additional difficulty that the mobile platform is often subject to nonholonomic constraints; one possibility is therefore to adapt kinematic [4] or optimal [5] control schemes. However, none of the above solutions is satisfactory when the objective is obstacle avoidance. For example, the optimal control formulation of the MPEP problem for mobile manipulators proposed in [6] leads to a nonlinear TPBVP whose solution can only be sought numerically, without any guarantee of success.

The objective of this paper is to present a family of probabilistic planners for solving the MPEP problem in mobile manipulators by extending our previous work dealing with the same problem in fixed-base redundant manipulators [7]. We exploit the natural partition of generalized coordinates between the manipulator and the mobile platform, and take into account the presence of nonholonomic constraints at the planning stage. All the planners rely on the same mechanism for generating random configurations that are compatible with the end-effector constraint.

The paper is organized as follows. In the next section, we give a precise formulation of the MPEP problem for mobile manipulators and clarify what we consider to be a solution. The procedure for generating random configurations is then presented, and the various proposed planners are described. Results for problems of increasing complexity are finally presented to illustrate the performance of the algorithms.

II. MPEP PROBLEM FOR MOBILE MANIPULATORS

In this section, we generalize our formulation of the MPEP problem [7] so as to apply to the mobile manipulator case. With respect to fixed-base manipulators, the essential features of mobile manipulators are the natural partition of generalized coordinates (mobile platform/manipulator) and the nonholonomy due to the rolling wheels.

Consider a mobile manipulator whose task is to move the end-effector along a given path in a workspace populated by obstacles. The direct kinematics is expressed as

\[ p = f(q) = f\left(\frac{q^p}{q^m}\right), \tag{1} \]

where \( p \in \mathbb{R}^3 \) is the end-effector pose (position and/or orientation) and \( q \in \mathbb{R}^N \) is the system configuration\(^1\), consisting of the platform configuration \( q^p \in \mathbb{R}^{N^p} \) and the manipulator configuration \( q^m \in \mathbb{R}^{N^m} \), with \( N^p + N^m = N \). While the manipulator subsystem is holonomic (i.e., arbitrary motions are possible for the manipulator configuration \( q^m \)), the motion of the platform is generated as

\[ \dot{q}^p = G(q^p)u, \tag{2} \]

\(^1\)We consider euclidean spaces for simplicity, but our developments apply to the case in which \( p, q^p \) and \( q^m \) are defined over manifolds.
where \( u \in \mathbb{R}^P \) are pseudoveLOCITIES (typically, linear and angular platform velocities), while the columns of \( G(q^p) \) span the null space of the nonholonomic constraint matrix.

An 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 \( \forall \sigma \in [0,1], p(\sigma) \in T \), where \( T \subset \mathbb{R}^M \) is the dextrous task space, defined as the set of end-effector poses that can be realized by \( \infty^{N-M} \) configurations. 

Assume that the mobile manipulator is kinematically redundant with respect to the given task, i.e., \( N > M \). Then, the MPEP problem is to find a configuration path \( \sigma \Rightarrow q(\sigma) = (q^p(\sigma), q^m(\sigma)) \) such that:

1. \( p(\sigma) = f(q(\sigma)), \forall \sigma \in [0,1] \);
2. the robot does not collide with obstacles or itself;
3. the path is feasible w.r.t. the kinematic constraints that may exist (e.g., manipulator joint limits);
4. the path is feasible w.r.t. the nonholonomic constraints, i.e., \( q^p(\sigma) \) is a solution of eq. (2).

Depending on the application, an initial joint configuration \( q(0) \) such that \( p(0) = f(q(0)) \) may or not be assigned. The first version of the problem is more constrained (and thus possibly easier to solve) than the second.

We seek a solution to the MPEP problem in the form of a sequence of configurations:

\[
\{ q(\sigma_0), q(\sigma_1), \ldots, q(\sigma_s) \}, \quad \sigma_0 = 0, \sigma_s = 1,
\]

with the \( \sigma_i \)'s equispaced and \( p(\sigma_i) = f(q(\sigma_i)) \). The integer \( s \) is called path sampling. A continuous path will be derived from this sequence by joining successive configurations by a local planner; this may use simple linear interpolation for \( q^m \), while the mobile platform nonholonomy must be taken into account when joining successive values of \( q^p \).

### III. Generation of Random Configurations

The algorithms we have developed for solving the MPEP problem share the same basic tool, i.e., a procedure which performs random sampling of self-motion manifolds. The mechanism proposed in [7] for generating \( N \)-dimensional random configurations compatible with the \( M \)-dimensional task constraint is based on a partition of \( q \) into \( M \) base and \( N - M \) redundant variables; the value of the latter is first randomly generated, and the value of base variables is then computed by inverse kinematics in such a way that the resulting configuration places the end-effector at a certain point of its assigned path. In this section, we show how this basic strategy can be adapted to mobile manipulators.

Assume for illustration that the platform is a unicycle, described by coordinates \( x, y, \theta \) (position and orientation) and controlled by pseudoveLOCITIES \( v, \omega \) (linear and angular velocity), while the manipulator is a spatial three-dof arm with rotational joints (Fig. 1). Hence, we have \( q^p \in \mathbb{R}^3 \), \( q^m \in \mathbb{R}^3 \), and \( q \in \mathbb{R}^6 \). The end-effector task is specified

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

3This will lead to path error between successive poses, whose entity can however be reduced at will by increasing the sampling \( s \).

4This sampling mechanism is similar to the one used in [8] for guaranteeing the closure constraint.

at the position level, i.e., \( p \in \mathbb{R}^3 \). The extension to the case where orientation is also specified is straightforward (provided that the manipulator has sufficient dof's).

As the configuration \( q \) is naturally split between platform variables \( q^p \) and manipulator variables \( q^m \), it is reasonable to choose the base/redundant partition accordingly. Since in our case \( M = 3 \) and \( N = 6 \), we must select three variables as redundant, and generate their values randomly. Throughout the paper, we use the \textit{platform} variables for this purpose. Other possibilities (e.g., using the manipulator variables or a mixed set) are not discussed here.

With the assumptions of Sect. II, each pose \( p(\sigma_i) \in \mathcal{T} \) along the given end-effector path can be realized by \( \infty^{N-M} \) configurations of the mobile manipulator, which represent the so-called self-motion manifold. Assume that the configuration \( q_i^p \) of the platform is randomly chosen. For each value of \( p_i = p(\sigma_i), i = 0, \ldots, s \), there exist a finite number (up to 4, in our case) of manipulator configurations \( q_m(i) = q_m(p_i, q_i^p) \) such that \( p_i = f(q_i^p, q_m(i)) \), computed by inverting the kinematic map (1) with \( q_i^p = q^p_i \). Depending on the value of \( q_i^p \), it may happen that no value of \( q_i^m \) is compatible with the end-effector pose \( p_i \).

According to the above strategy, the procedure generating a random sample of the self-motion manifold corresponding to \( p_i \) is described in pseudocode as follows:

```plaintext
RAND_CONF(p_i, q_bias)
q^p_i = RAND_PLATFORM(q^p_bias)
q^m_i = INV_KIN(p_i, q^p_i, q^m_bias)
if INV_KIN_FAIL
   Return RAND_CONF_FAIL
else Return q_i = (q^p_i, q^m_i)
```

RAND_PLATFORM(q^p_bias)

```plaintext
u_i = RAND_INP(q_bias)
q^p_i = MOVE(q^p_bias, u_i)
Return q^p_i
```

The effect of the optional argument \( q_{bias} \), which appears in both procedures, is to bias the distribution characterizing the randomly generated samples. When \( q_{bias} \) is present, RAND_CONF returns (if successful) a configuration \( q_i \) such that \( (i) \) there is a feasible path connecting \( q^p_{bias} \) to \( q^p_i \), \( (ii) \) \( p_i = f(q_i) \), and \( (iii) \) \( \|q^m_i - q^m_{bias}\|_\infty < d \), where \( d \) is a maximum allowed joint displacement. The first of these properties is guaranteed by RAND_PLATFORM, which generates a random pseudovelocicty vector \( u_i \) and then computes (by forward integration of eq. (2)) the platform

5To be precise, the inverse image of any point \( p \in \mathcal{T} \) is in general a finite number of disjoint manifolds.
A more accurate computation is possible following the ideas in [9].
problem for mobile manipulators. All of them make use of the collision checking procedure NO_COLL. When invoked with a single argument \( q_i \), it performs a collision check (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 path joining them (by sampling it at a sufficiently high rate). In particular, a feasible path (produced under the action of the constant pseudovelocity inputs selected by RAND_INP) is used for for the mobile platform, while the manipulator path is obtained by linear interpolation.

A. Greedy Planner

The core of the first algorithm is the STEP function which, given two generic poses \( p_i, p_k \) \((0 \leq i < k \leq s)\) belonging to the end-effector sequence and a configuration \( q_j \) on the self-motion manifold of \( p_i \), builds 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.

\[
\text{STEP}(i, p_i, q_j, k) \\
\text{for } j = i \text{ to } k - 1 \text{ do} \\
\quad \text{while } l < \text{MAX SHOTS and } !\text{Succ} \text{ do} \\
\quad\quad q_{j+1} \leftarrow \text{RAND CONF}(p_{j+1}, q_j); \\
\quad\quad \text{if } !\text{RAND CONF FAIL and NO COLL}(q_j, q_{j+1}) \\
\quad\quad \quad \text{Succ} \leftarrow 1; \text{ADD TO PATH}(q_{j+1}); \\
\quad\quad \quad l \leftarrow l + 1; \\
\quad\quad \text{if } l = \text{MAX SHOTS} \\
\quad\quad \quad \text{Return STEP FAIL} \\
\quad\quad \text{else} \\
\quad\quad \quad j \leftarrow j + 1; \\
\quad \text{Return PATH} \\
\]

The parameter 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 \( p_{j+1} \), sufficiently close to the bias configuration \( q_j \), and such that the path between \( q_j \) and \( q_{j+1} \) is feasible, the whole 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 is to devise a greedy algorithm based on iterated calls to the STEP function with \( p_0, p_s \) as subsequence extrema and random \( q_0 \).

GREEDY algorithm
\[
j \leftarrow 0; \\
\text{while } j < \text{MAX ITER} \text{ and } \text{STEP FAIL} \text{ do} \\
\quad q_0 \leftarrow \text{RAND CONF}(p_0); \\
\quad \text{STEP}(0, p_0, q_0, s); \\
\quad j \leftarrow j + 1; \\
\quad \text{if } !\text{STEP FAIL} \\
\quad \quad \text{Return PATH} \\
\quad \text{else} \\
\quad \quad \text{Return FAILURE} \\
\]

Given the initial pose \( p_0 \), RAND_CONF\((p_0)\) generates an initial configuration \( q_0 \) as described in the previous section. STEP is then invoked to search for a sequence of configurations guaranteeing feasible collision-free motion while the end-effector moves from \( p_0 \) to \( p_s \). In case of success, the path found by STEP is returned. If STEP fails and MAX ITER has not been exceeded, a new \( q_0 \) is generated and STEP starts a new search from \( q_0 \).

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, and discarded if STEP does not reach the last self-motion manifold. Experiments have shown that this planner is effective in easy problems (see Sect. V), 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 complex problems.

B. RRT-Like Planner

To overcome the limitations of the depth-first algorithm GREEDY, one may try to generate multiple random samples for each self-motion manifold and to connect configurations on successive manifolds by local paths. As in [7], this exploratory behavior is achieved by the RRT LIKE algorithm, which adapts the notion of RRT (Rapidly-exploring Random Tree, [11]) to mobile manipulators.

Our algorithm tries to expand a tree \( \tau \) 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 path is extracted by graph search techniques.

RRT LIKE algorithm
\[
\text{RRT LIKE} \\
\text{algorithm} \\
\quad j \leftarrow 0; \\
\quad \text{while } p_{new} != p_s \text{ and } j < \text{MAX ITER} \text{ do} \\
\quad\quad q_0 \leftarrow \text{RAND CONF}(p_0); \\
\quad\quad \text{CREATE}(\tau, q_0); \\
\quad\quad i \leftarrow 0; \\
\quad\quad \text{repeat} \\
\quad\quad\quad p_{new} \leftarrow \text{EXTEND LIKE}(\tau); \\
\quad\quad\quad i \leftarrow i + 1; \\
\quad\quad\quad \text{until } p_{new} = p_s \text{ or } i = \text{MAX EXT} \\
\quad\quad j \leftarrow j + 1; \\
\quad\quad \text{if } p_{new} = p_s \\
\quad\quad \quad \text{Return } \tau \\
\quad\quad \text{else} \\
\quad\quad \quad \text{Return FAILURE} \\
\quad \text{END EXTEND LIKE} \\
\quad \text{END RRT LIKE} \\
\]

EXTEND LIKE\((\tau)\)
\[
q_{rand} \leftarrow \text{RAND CONF}; \\
(q_{near}, k) \leftarrow \text{NEAR NODE}(q_{rand}, \tau); \\
q_{new} \leftarrow \text{RAND CONF}(p_{k+1}, q_{near}); \\
\text{if } !\text{INV KIN FAIL and NO COLL}(q_{near}, q_{new}) \\
\quad \text{ADD NODE}(\tau, q_{new}); \\
\quad \text{ADD EDGE}(\tau, q_{near}, q_{new}); \\
\quad \text{Return } p_{k+1} \\
\text{else} \\
\quad \text{Return NULL} \\
\]

First, RAND_CONF is called with no arguments to find a random \( q_{rand} = (q_{rand}^p, q_{rand}^m) \), and NEAR_NODE
identifies \( q_{\text{near}} \), the node of \( \tau \) closest to \( q_{\text{rand}} \) with respect to the platform variables, and returns the index \( k \) of the end-effector pose \( p_k \) to which \( q_{\text{near}} \) is associated. Then, RAND_CONF computes \( q_{\text{new}} \) by generating first a random input (using one of the strategies described in Sect. III-B), then the corresponding \( q_{\text{new}}^\tau \), starting from \( q_{\text{near}} \), and finally (if successful) \( q_{\text{new}}^\tau \) by inverse kinematics on the manifold associated to \( p_{k+1} \). The path joining \( q_{\text{near}} \) to \( q_{\text{new}} \) is now checked for collision; if the result is negative, \( \tau \) is expanded and \( p_{k+1} \) is returned. Figure 3 shows the expansion of the tree \( \tau \) in the subspace of platform configurations.

The role of \( q_{\text{rand}} \) in guiding the expansion of RRT\_LIKE is only to identify \( q_{\text{near}} \), the closest node to \( q_{\text{rand}} \) w.r.t. the platform variables; the direction of expansion from \( q_{\text{near}} \) is then determined by the choice of the pseudovelocity input, and in general does not depend on \( q_{\text{rand}} \). This is quite different from what happens in the classical RRT algorithm, where the direction of expansion is chosen to be the line joining \( q_{\text{rand}} \) with \( q_{\text{near}} \). To retain this strategy, which is indeed essential for the effectiveness of RRT (namely, for driving the expansion toward wide Voronoi regions), one can adopt the optimal pseudovelocity generation outlined in Sect.III-B, using the \( I_{\text{dist}} \) criterion with \( q_{\text{des}} = q_{\text{rand}} \).

C. Variations on RRT\_LIKE

The expansion of \( \tau \) toward randomly selected directions gives to RRT\_LIKE an exploratory attitude which, for the MPEP problem, could prove inefficient due to the strong constraint represented by the end-effector path. However, it is 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; the arguments passed to STEP are \( p_{1}, p_{s} \), where \( p_{1} \) is the closest pose to \( p_{s} \) reached so far by the algorithm. This modified RRT-based planner, which tries 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, is called RRT\_GREEDY.

The exploratory attitude of RRT\_LIKE may also be a drawback when the start and goal self-motion manifolds are very distant. Inspired by [12], one possible solution is to expand two trees, respectively rooted at the start and at the goal self-motion manifolds. The trees will ‘meet’ on some intermediate manifold, but different nodes will be generated; it is then necessary to compute a self-motion connecting the two nodes by an RRT-based search restricted to the manifold. This planner is called RRT\_BIDIR.

V. PLANNING EXPERIMENTS

We now present some MPEP experiments for the mobile manipulator of Fig. 1. The algorithms were implemented in C on a 1 Ghz PC and integrated in the software platform Move3D, dedicated to motion planning and developed at LAAS-CNRS, France\(^8\).

The first two experiments aim at highlighting the effects of the different pseudovelocity generation procedures of Sect.III-B: to this end, the mobile manipulator must move its end-effector along a given rectilinear path in the absence of obstacles, and only the RRT\_LIKE planner is used.

For the first experiment, Fig. 4 compares the solution tree (more precisely, its projection on the \( x, y \) plane) obtained by completely random pseudovelocity inputs (left) and constant-energy optimal inputs with criterion \( I_{\text{dist}} \) (right). The left tree is uselessly erratic (given that no obstacles are present) if compared with the right one, which extends mostly along the direction of the task trajectory. The table reports a performance comparison (averaged over 20 trials) between these two and other methods, in terms of time needed to find a solution, failures of INV\_KIN, and nodes in the tree. The superiority of constant-energy methods is confirmed by the smaller number of kinematic inversion failures and by the reduced tree size. As for the choice of the performance criteria, the minimization of \( I_{\text{dist}} \) appears to be more effective than the maximization of \( I_{\text{comp}} \). \( I_{\text{mix}} \) denotes a weighted criterion that combines the two.

The second experiment is similar to the first, but a different initial configuration has been assigned. In particular, the arm is completely stretched and the mobile platform orientation is orthogonal to the end-effector path; hence, \( q_{0} \) has a low task compatibility with respect to the given end-effector trajectory. Figure 5 shows the results obtained by RRT\_LIKE using constant-energy optimal inputs with the performance criteria \( I_{\text{dist}} \) (left) and \( I_{\text{mix}} \) (right). In this case, the use of \( I_{\text{comp}} \) allows the robot to recover and maintain a higher compatibility value (note the absence of kinematic inversion failures), ultimately resulting in a smoother motion and in a smaller computation time.

Experiments of the second group take place in environments with obstacles, and aim at comparing the performance of the various planners. In these trials, pseudovelocity inputs are generated by optimizing the mixed performance criterion \( I_{\text{mix}} \) over a set of four constant-energy candidates in \( U_{1}, \ldots, U_{4} \) (see Sect. III-B). As before, the planners’ performances are averaged over 20 trials.

The third experiment scene is quite simple. The robot must move its end-effector along a polynomial path through
Pseudovelocity generation | time (s) | # kin fail | # nodes
---|---|---|---
completely random | 2.16 | 41 | 165
constant energy | 0.50 | 17 | 127
optimal (comp. rand., $I_{\text{dist}}$) | 2.00 | 59 | 158
optimal (const. ene., $I_{\text{dist}}$) | 0.33 | 4 | 96
optimal (const. ene., $I_{\text{comp}}$) | 1.00 | 10 | 189
optimal (const. ene., $I_{\text{mix}}$) | 0.60 | 21 | 115

Fig. 4. First experiment: The tree built on the $x$, $y$ plane by RRT\_LIKE using completely random inputs (left) and constant-energy optimal inputs with criterion $I_{\text{dist}}$ (right). Also reported is a comparison extended to other input generation methods.

two columns. Figure 6 contains some frames from the solution path computed by GREEDY and a table summarizing the performance of the planners. In this case, the GREEDY planner performs best under all aspects due to its depth-first strategy, which is invariably more effective in easy planning problems. Also RRT\_BIDIR achieves a good result thanks to the large space available for reconfiguration.

Figure 7 shows the scene of the fourth experiment; the end-effector must follow a rectilinear path which is dangerously close to an obstacle. The solution shown was planned by RRT\_GREEDY; note how the robot stretches the arm to move under the obstacle. RRT-based algorithms perform best in this case thanks to their exploratory attitude, with the exception of RRT\_BIDIR, which is penalized by the reduced space for reconfiguration in the contact manifold.

The fifth planning problem is very difficult: to complete its task, the mobile manipulator must first cross a narrow passage and then carefully move its arm so as to drive the end-effector between the sandwich-shaped obstacle. While GREEDY failed to produce a solution within the allotted time, RRT-based planners performed quite well.

A number of experiments have confirmed the above indications. Essentially, GREEDY is very effective when dealing with simple queries, while RRT\_LIKE and RRT\_GREEDY perform much better when the difficulty of the problem increases. The bidirectional strategy of RRT\_BIDIR is convenient when there is a large space available for reconfiguration, as in the case of Fig. 9.

VI. CONCLUSIONS

Single-query probabilistic planners have been presented for the problem of generating collision-free motions for a nonholonomic mobile manipulator moving along a given end-effector path. Experiments show that simple instances of the problem can be solved more efficiently by a greedy approach, whereas the breadth-first search of RRT-based planners is needed to deal with more complex cases.

Among the issues deserving further attention, we mention (i) proving probabilistic completeness along the lines of [8,11], (ii) complexity analysis, and (iii) the choice of performance criteria for pseudovelocity generation, with the ideas in [13] as possible inspiration. An extension of the proposed methods to sensor-based exploration can also be envisaged following the approach in [14].
### REFERENCES


---

<table>
<thead>
<tr>
<th>Planner</th>
<th>time (s)</th>
<th># cc</th>
<th># nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>GREEDY</td>
<td>12.6</td>
<td>478</td>
<td>40</td>
</tr>
<tr>
<td>RRT_LIKE</td>
<td>6.3</td>
<td>218</td>
<td>115</td>
</tr>
<tr>
<td>RRT_GREEDY</td>
<td>33</td>
<td>1230</td>
<td>511</td>
</tr>
</tbody>
</table>

Fig. 7. Fourth experiment: Solution obtained with RRT_GREEDY and comparison of planners’ performance.

<table>
<thead>
<tr>
<th>Planner</th>
<th>time (s)</th>
<th># cc</th>
<th># nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>GREEDY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RRT_LIKE</td>
<td>47</td>
<td>5439</td>
<td>229</td>
</tr>
<tr>
<td>RRT_GREEDY</td>
<td>230</td>
<td>15769</td>
<td>1322</td>
</tr>
</tbody>
</table>

Fig. 8. Fifth experiment: Solution obtained with RRT_LIKE and comparison of planners’ performance.

<table>
<thead>
<tr>
<th>Planner</th>
<th>time (s)</th>
<th># cc</th>
<th># nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>GREEDY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RRT_LIKE</td>
<td>33</td>
<td>1768</td>
<td>560</td>
</tr>
<tr>
<td>RRT_GREEDY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RRT_BIDIR</td>
<td>278</td>
<td>115</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 9. A problem for which RRT_BIDIR is more efficient than other planners.