Workspace-Guided Rapidly-Exploring Random Tree Method for a Robot Arm

Jaesik Choi
choi31@cs.uiuc.edu
August 12, 2007

Abstract

Motion planning for robotic arms is important for real, physical world applications. The planning for arms with high degree-of-freedom (DOF) is hard because its search space is large (exponential in the number of joints), and the links may collide with static obstacles or its joints (self-collision). In this paper we present a factor-guided sampling based motion planning algorithm that finds plans of motion from one arm configuration to a goal arm configuration in 2D and 3D space. Our algorithm finds a workspace-based roadmap with utilizing the locations of end-effector. Then, with the roadmap, a sampling based motion planner (Rapidly-Exploring Random Tree [1]) finds a configuration space (CSpace) based roadmap. The RRT is operated as a single-query mode which have no pre-planned roadmap. Our algorithm is unique in two ways: (a) it takes only polynomial time in the number of joints to find a workspace-based roadmap; and (b) it utilizes the topology of the arm and obstacles to factor the search space reduce the complexity of the planning problem using dynamic programming. Thus, our algorithm dramatically reduce the time to find a path between a start configuration and a goal configuration. The experimental results show that the proposed algorithm improves the performance of path planning for 2D.

1 Introduction

Robot motion planning focuses on finding paths from one robot configuration to another. Robotic arms are particular robots that are made of connected links and joints (their degrees-of-freedom (DOF)). They are used for general-purpose manipulation of the work space, and planning with them is hard because of traditionally high DOFs. Sampling-based methods and Grid-based methods are proposed to solve the problem [1, 2, 3, 4].

The motion planning problems for multi-DOFs arms are still the hard, although sampling-based planners [1, 2] easily solve many intractable problems in the motion planning literature. The problems are hard due to the three reasons. (1) When

1This paper is the final report for a course of University Illinois, ECE550 which is taught by Prof. Seth Hutchinson.
the number of joints increase, it quickly causes the intractable configuration space (CSpace) whose complexity is exponentially proportional to the number of joints. (2) Current single-query sampling-based methods (Expansive Space Trees (ESTs) and Rapidly-Exploring Random Trees (RRTs) [1]) focus on expanding tree toward all directions or toward a goal direction. However, the planner has trouble due to a curse-of-dimension of CSpace when expanding toward all directions. Moreover, the planner is easily stuck by local minima, when expanding toward a goal direction. (3) Most importantly, the CSpace only planners are biased on the locations whose configurations are heavily redundant. In a low-DOFs arm, there is a few redundant configurations which indicate the same location in end-effector. However, in a high-DOFs arm, many redundant configurations indicate the same location. Thus, the sampled configurations in CSpace are heavily biased on some workspace locations which have many redundant configurations.

Workspace of the arms can be used to reduce the complexity of the planning problems. Some previous researches (eg. [4]) use balls or polygons which include the robot body. The balls or polygons reduce computation in planner, because they have simpler body shape than the original robot. However, this method is not appropriate for the high-DOFs arms in the cluttered environments, because they need many balls or polygons to represent the arms in such environments.

In this paper, we present a motion planning algorithm which uses the workspace roadmap to reduce the complexity of planning problem. The workspace roadmap is the potentially possible trajectory of the end-effector. Our algorithm is composed of two parts: the preprocessing part and the planning part.

The preprocessing part finds (1) all the reachable locations of the end-effector, (2) all the valid actions for each workspace location, and (3) all the redundant configuration for the location. The kinematics of robotic arm is given to the preprocessing algorithm. It is done with a dynamic programming which iterates from the innermost joint to the outermost joint within polynomial time to the number of joints (The detail algorithm is described in 3.1).

The planning part is also composed of two parts: a workspace roadmap planner, and a workspace-guided RRT planner. The workspace roadmap planner finds a workspace roadmap given the problem (an initial configuration and a goal configuration). It finds a valid trajectory of end-effector (workspace roadmap) with a modified A* algorithm using the valid actions which is found by the preprocessing algorithm. The RRT algorithm uses the workspace roadmap to guide the sampling. At first, the goal configuration of RRT is the next configuration of initial configuration in the trajectory. When the location is once occupied (or found) by the RRT, the RRT change the goal configuration with the next configuration in the trajectory.

Our algorithm is an efficient single-query sampling-based planner when we compare to the previous RRT planners. It dramatically reduce the search space and complexity of the planning problem. Moreover, the preprocessing only takes the polynomial time to the number of joints. The previous of this research [5] has similar shape of algorithm. However, it assume that there is no self-collision between inner links. Moreover, there is no experimental comparison between the grid-based A* algorithms and the probabilistic roadmap methods.

Section 2 shows the motivational example of this research. Section 3 present the
algorithm of this planning algorithm.

2 Motivations

We are interested in finding sets of plans that moves the position of end-effector (eg. an action from \( p_0 \) to \( p_1 \) in Figure 1(R1)). The algorithm first finds the set of accessible positions (eg. \( p_0 \)) for joint 1 and their neighboring positions (eg. \( p_1 \) and \( p_2 \)). For each of those positions it marks the possible entrance configurations, any angle with which link 1 can approach from joint 1 to joint 2 at that position (in this case, the location of joint 1 is fixed, so there is at most one such angle for every position).

The algorithm proceeds in a dynamic-programming fashion as follows. For each position for joint 2, we find the set of accessible positions. Each position for joint 2 (eg. \( l_0 \) in Figure 1(R2)) has reachable positions (eg. \( l_1 \), \( l_2 \), \( l_3 \), and \( l_4 \)). Some (eg. \( l_2 \) and \( l_3 \)) are reachable by moving the current joint. The others (eg. \( l_1 \) and \( l_4 \)) are found by using the plans of the previous joint. For example, the \( l_0 \) moves to \( l_1 \), when the \( p_0 \) in joint 1 moves to \( p_1 \). That is, a plan (from \( l_0 \) to \( l_1 \)) is built by the plan of the joint 1 (from \( p_0 \) to \( p_1 \)).

Our algorithm takes care the topology of the arm. In Figure 1 (M1), a position has two entrance configurations: \( \text{conf}_l \); and \( \text{conf}_r \). If the two configurations have the same topology, it merges the two sets of plans and selects one representative (eg. \( \text{conf}_l \)). However, in Figure 1 (M2), it maintains the separated sets of plans, because their topologies are difference due to an obstacle.

For each possible position for joint 2 we find the set of accessible positions for joint 3. Every positions for joint 3 may have many entrance configurations as well, so we mark those configurations with a set of segments. Similarly, we proceed for joint 4 and 5.

We have all the possible actions for each end-effector position, after the dynamic program reach to the last joint (\( m \)) of the arm. Thus, workspace roadmap is easily built given the start configuration and the goal configuration (eg. a wavefront expansion method [6] as seen in the Figure 2. Here, the detail configurations are not considered except the topology of the arm against the obstacles. The found path is the workspace roadmap which is the desirable trajectory of the arm to reach to the goal configuration.

The each position in the workspace is the subgoal of the RRT algorithm. That is, the just next position of the start configuration is given to the RRT algorithm. When RRT finds a path to the position, the following positions in the trajectories are given to the RRT algorithm. When RRT finds a path to the goal position, the RRT stops to expand.

As seen Figure 3, we maintains a partially ordered tree to represent a segment, assuming that there is always a path between two configurations in the same segment. The partially ordered tree can be represented with tractable number of vertexes and edges, although the all the possible configurations in a segment (Closed Kinematic Chain) can be large (\( O(c^{m-2}) \)) (when \( c \) is a constant, and \( m \) is the number of joints). \(^1\)

\(^1\)In some cases, the planning within the segment is much easier than the planning in the whole configuration space.
Figure 1: (R1)/(R2) shows the reachable locations with actions from an end-effector location (respectively \(p_0, l_0\)). (M1)/(M2) shows the condition of merging two sets of actions. (R1) represents a location which is reachable by a link and a joint. The two neighboring locations are reached by moving the joint 1 toward left or right. (R2) represents a position which is reachable by two links. Four neighboring positions (\(l_1, l_2, l_3,\) and \(l_4\)) are respectively lead by 4 actions: moving joint 1 toward the left (\(l_1\)); moving joint 2 toward the left (\(l_2\)); moving joint 2 toward the right (\(l_3\)); and moving joint 1 toward the right (\(l_4\)). (M1) shows the two set of configurations and their actions which can be merged into a group. (M2) shows the two separated set of configurations and their actions which cannot be merged because they have different topology due to an obstacle.

Figure 2: Figures from (1) to (6) show the sequence of planning. The figures focus only on each step of the plan, although our algorithm finds all the possible paths with breadth-first search. (1) is the planning task from an initial configuration to a goal one. There is no direct path for the task because the obstacle splits two types of topologies of the arm (left sides and right sides). In each step (2)(3)(4)(5), our algorithm provides neighboring positions from the current position. The goal position is found after searching the 2D workspace which limits search space.
Figure 3: This figure represents the partially ordered tree representation for all the possible configurations. The arm has 6 joints from J0 to J5 (J6 is the end-effector of the arm). In this example, the 1st, the 3rd and the 5th joints are have two locations. Thus, all the possible locations to manage is \( \frac{n}{2} \times 3 \) (=9). However, all the possible configuration is \( 2^n \) (=8) which is exponentially proportional to the number of joints. Thus, it easily become intractable, when the number of joints increase. Our algorithm builds the partially ordered tree (ES) in the preprocess workspace algorithm.
3 Algorithm

The Workspace-Guided RRT in algorithm 1 is composed of two parts: the preprocessing part and the planning part.

The preprocessing part (PreprocessWorkspace in algorithm 2) finds all the possible locations and actions of the arm in each possible configuration.

The planning part has two subparts: FindWorkspaceRoadmap in algorithm 3; and SubGoalRRT in algorithm 4. Given the initial configuration and goal location, FindWorkspaceRoadmap finds a possible trajectory of the end-effector (abstract path). Then, FindWorkspaceRoadmap finds a concrete path given the trajectory of the end-effector. That is, the locations of end-effector become a sub-goal of FindWorkspaceRoadmap.

Algorithm: Workspace-Guided RRT Algorithm

**Input:**
- $c_{\text{start}}$: the start configuration
- $c_{\text{goal}}$: the goal configuration
- $\{\text{len}_i\}_{i \leq m}$: the lengths of pegs
- $\text{obs}$: Obstacles

**Output:**
A RRT which has the path from $c_{\text{start}}$ to $c_{\text{goal}}$.

```plaintext
(Act, ES) ← PreprocessWorkspace(\{\text{len}_i\}_{i \leq m}, \text{obs})

/* Act: sets of possible actions for each location */
/* ES: sets of all configuration for each location */
W_{\text{roadmap}} ← FindWorkspaceRoadmap(Act, ES, c_{\text{start}}, c_{\text{goal}})

c_{\text{current}} ← c_{\text{start}}

foreach location of end-effector(l) ∈ W_{\text{roadmap}} from the start to goal do
    $c_{\text{next}} ← \text{FindClosestNext}(c_{\text{current}}, l, ES)$
    $\langle \text{RRT}, c_{\text{current}} \rangle ← \text{SubGoalRRT}(\text{RRT}, c_{\text{next}}, \text{obs})$

return RRT
```

3.1 Preprocessing the Workspace

PreprocessWorkspace finds all the reachable locations of the end-effector. It also finds all the possible actions for a location of end-effector. Here, an action is a movement of an inner joint in the configuration. An action is composed of three elements: the current location (and orientation) of the end-effector; the next location (and orientation) of the end-effector; and the location of pivot joint. Thus, this algorithm simplifies the motion planning problem (in the configuration space) into the relocation problem of end-effector (in the workspace).

It iterates from the inner most joint to the outer most joint (end-effector) with the dynamic programming fashion. It reduces the space by merging two configurations, if their configurations are homotopy $^2$.

$^2$Here, we call that two configurations are homotopy, if they indicate the same end-effector and they are...
Algorithm 2: PreprocessWorkspace finds all the reachable locations of the end-effector. It also finds all the possible actions for a location end-effector. Here, an action is a movement of any inner joints in the configuration. It iterates from the inner most joint to the outer most joint (end-effector) with the dynamic programming fashion. It reduces the space by merging two configurations, if their configurations are homotopy in the workspace.
3.2 Planning

The planning part is composed of workspace roadmap and the workspace-guided RRT.

3.2.1 A Workspace Roadmap Planner

_FindWorkspaceRoadmap_ finds the trajectory of the end-effector given an initial configuration and a goal location. We provide the location of end-effector instead of goal configuration. This is reasonable problem, because general tasks for robotic arms are more focused on the end-effector.

The _PreprocessWorkspace_ pass all the possible actions to _FindWorkspaceRoadmap_. Given the possible reachable locations and possible actions (movements) among them, it finds a valid path with a wavefront expansion-like algorithm. Thus, it returns a valid trajectory of the end-effectors.

It starts from the initial location (and orientation) of end-effector. Then, it finds all the neighboring locations in terms of a unit action. That is, it finds all the locations which can be reached by a unit action from the initial location. It iterates this process until it reaches to the goal location.

The returned path is the trajectory of end-effector, because we simplify the configuration space into the work space in the preprocessing algorithm. However, the trajectory is an important roadmap, because it merges configurations which have similar topology against obstacles.

3.2.2 A Workspace-Guided RRT Planner

_SubGoalRRT_ is the same shape with another RRT algorithm. It construct a tree from the initial configuration to the goal configuration. In some case, it builds a tree without specific problem (initial and goal) to analyze the structure of configuration.

The root of tree locates on the initial configuration. It generates a sample in each iteration. For the sample, it finds the nearest node in the tree. Then, it makes a branch from the node to the sample with a certain length. It iterates until it reach to the goal configuration.

We use this _SubGoalRRT_ to find concrete paths between two adjacent points in the workspace roadmap (A planned trajectory of end-effector).

4 Experiments

In experiments, we use high-DOFs (20 DOFs or 30 DOFS) 2D arms. We compare the performance (planning time in seconds) in three algorithms: A* algorithm only (A*); A* algorithm + Workspace Roadmap (A* + WR); and RRT + Workspace Roadmap (RRT + WR). We also compare the performance with the assumption of self-collision and without the assumption (of self-collision).

continuously deformed each other.
Algorithm: FindWorkspaceRoadmap

Input:
- \(Act\): sets of possible actions for each location
- \(ES\): sets of all configuration for each location
- \(c_{start}\): the start configuration
- \(c_{goal}\): the goal configuration

Output:
- \(W_{roadmap}\): workspace-based roadmap

/* A modified WaveFront expansion in workspace */

\[
\begin{align*}
\text{l}_{\text{start}} & \leftarrow \text{end-effector of } c_{\text{start}} \\
\text{l}_{\text{goal}} & \leftarrow \text{end-effector of } c_{\text{goal}} \\
\text{list}_{\text{act}} & \leftarrow \text{l}_{\text{start}} \\
\text{list}_{\text{visit}} & \leftarrow \phi \\
\text{reached} & \leftarrow \text{false} \\
\text{while} & \text{not reached do} \\
& \text{foreach } \text{l} \in \text{list}_{\text{act}} \text{ and } \text{l} \notin \text{list}_{\text{visit}} \text{ do} \\
& \hspace{1em} \text{foreach } \text{l}_{\text{next}} \in \text{Act(l)} \text{ and } \text{l}_{\text{next}} \notin \text{list}_{\text{visit}} \text{ do} \\
& \hspace{2em} \text{list}_{\text{next}} \leftarrow \text{list}_{\text{next}} \cup \{\text{l}_{\text{next}}\} \\
& \hspace{2em} \text{plan} \leftarrow \text{plan} \cup \{\text{directed edge(l, l}_{\text{next}})\} \\
& \hspace{2em} \text{if } \text{l}_{\text{next}} = \text{l}_{\text{goal}} \text{ then } \text{reached} \leftarrow \text{true} \\
& \text{list}_{\text{act}} \leftarrow \text{list}_{\text{next}} \\
& \text{list}_{\text{next}} \leftarrow \phi \\
\text{return} & \text{shorted path in plan from l}_{\text{start}} \text{ to l}_{\text{goal}}
\end{align*}
\]

Algorithm 3: FindWorkspaceRoadmap finds the trajectory of the end-effector given an initial configuration and a goal location. The PreprocessWorkspace pass all the possible actions to FindWorkspaceRoadmap. Given the possible reachable locations and possible actions (movements) among them, it finds a valid path with the wavefront expansion like way. Thus, it returns a valid trajectory of the end-effectors.
Algorithm: SubGoalRRT

Input:
\( \text{RRT}(V, E) \): Currently built Rapidly-Exploring Random Tree
\( c_{\text{next}} \): the next goal configuration
\( \text{obs} \): obstacles

Output:
\( \text{RRT} \): Expanded RRT
\( c_{\text{current}} \): currently achieved configuration

/* \( c_{\text{next}} \) (a subgoal) directed RRT algorithm */

\( \text{reached} \leftarrow \text{false} \)

while not \( \text{reached} \) do

\( q_{\text{rand}} \leftarrow \) a randomly chosen from \( c_{\text{next}} \) (90%) and other free configurations (10%)

\( q_{\text{near}} \leftarrow \) closest neighbor of \( q_{\text{rand}} \) in \( \text{RRT} \)

\( q_{\text{new}} \leftarrow \) progress \( q_{\text{near}} \) by step size along the straight line

if \( q_{\text{new}} \) is collision free then

\( V \leftarrow V \cup \{q_{\text{new}}\} \)

\( E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\} \)

if \( (\text{end-effector of } q_{\text{new}}) = (\text{end-effector of } c_{\text{next}}) \) then

\( \text{reached} = \text{true} \)

return \( (\text{RRT}, q_{\text{new}}) \)

Algorithm 4: SubGoalRRT finds a concrete path from the current configuration to a sub goal location. It constructs a tree in the configuration space from the initial configuration to the goal configuration. It generates a random sample and connects the sample with the nearest node in the tree. If the tree reach the goal location, it return the found concrete path.

The RRT methods need a good subroutine to find the nearest neighbor. However, the size of tree in the configuration space easily become huge. Then, the naive (nearest neighbor) methods takes long time to find the nearest points in the tree given the sampled point. Many nearest neighbor algorithms measure the distance of two configuration in the Euclidian space. However, it is a huge burden in the high dimensional configuration space.

Thus, we adopt a technique of LSH (Locally Sensitivity Hashing) [7] by using the locations of certain joints. We project the configuration in the configuration space into the point in the smaller space. That is, we use locations of some inner joints to compare the distance between two configurations. It takes much less time than the linear search.

As an A* algorithm, we use ARA algorithm [3] which is one of the fastest grid-based A* algorithm.

4.1 Experimental Environment

Figure 4 shows the experimental settings and the planned paths (no self-collision cases). The settings have various environments including cluttered environments (A, C), a small hole problem (B), and a floating obstacle problem (D).

In Figure 5, we marks as ‘-‘ when the setting does not return any result until the test machine runs up the all memory. It takes 900 secs to reach the level.
4.2 Experimental Result

Figure 5 shows that the results with the assumption of no self-collision outperform the results with the self-collision. The planning in self-collision is harder than planning without self-collision, as we expect. Especially, (B) (with a narrow hole) has no result with the self-collision settings in the reasonable time (900 secs).

The A* algorithm is better than the workspace roadmap-based RRT (RRT + WR) in settings (B, C). However, A* algorithm has no result in (A), although the RRT algorithm finds solutions.

However, Workspace-based A* algorithm outperforms the others in all cases. It also finds solutions in all self-collision settings, although it takes longer time than the settings with no self-collision. Especially, there is no solution other than the workspace-based A* in (D) with a floating obstacle.

The RRT algorithm is originally designed not for the specific problem but for constructing the skeleton of configuration space. The RRTs would be better to find the global structure. Thus, it is little bit unfair to compare the performance for the specific tasks (given a goal location). In most case, original RRT algorithms takes much longer time than the goal-directed (workspace-guided) RRTs.

5 Conclusion

In this research, we combine a probabilistic roadmap algorithm and a workspace-based roadmap algorithm. For the goal-direct problems, the combined algorithm shows the better performance than the general-purpose RRT algorithm. The workspace-guide A* algorithm also shows much better performance than the general-purpose A* algorithm.
Figure 5: The table shows the experimental result (planning time in seconds) with various settings (A), (B), (C), and (D). We compare the performance with various algorithms: ARA (A*); ARA + Workspace Roadmap (A* + WR), Workspace Roadmap + RRT (RRT + WR) with x% sampling. Here, the (100 - x)% of points are sampled in the subgoal. The x% of points are freely sampled. We use two different parameters: 10% and 30%. The '-' means that there is no result within the reasonable time (900 sec). Due to the lack memory (3GB), the test machine cannot continue experiments anymore.

<table>
<thead>
<tr>
<th>Environments</th>
<th>w/o self-collision</th>
<th>w/ self-collision</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A*</td>
<td>A* + WR</td>
</tr>
<tr>
<td>(A)</td>
<td>-</td>
<td>89</td>
</tr>
<tr>
<td>(B)</td>
<td>83</td>
<td>5</td>
</tr>
<tr>
<td>(C)</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>(E)</td>
<td>-</td>
<td>9</td>
</tr>
</tbody>
</table>

References


