2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

PROT: Productive Regions Oriented Task space path planning for hyper-redundant manipulators

Junghwan Lee[1] Sung-eui Yoon[2]

Abstract— **In this paper we propose a novel efficient sampling bias technique to improve the performance of a task space trajectory planner for hyper-redundant manipulators. We defines productive regions in the task space as a set of states that can lead effectively to a goal state. We first compute a maximum reachable area (MRA) where a robot can reach from the node by an employed local planner for a node in the task space. When the MRA of a node contains the goal state, we call it *promising* and bias our sampling to cover promising MRAs. When the MRA does not contain the goal state, we call it *unpromising* and construct a detour sampling domain for detouring operations from obstacles constraining the manipulator. The union of promising MRAs and detour sampling domains approximates our productive regions, and we bias our sampling to cover these domains more. We have applied our Productive Regions Oriented Task space planner (PROT) to various types of robots in $\mathbb{R}^2$ task space and achieved up to 3.54 times improvement over the state-of-the-art task space planner. We have additionally verified the benefits of our method by applying it to cabled mobile robot planning.**

I. INTRODUCTION

The trajectory planning problem for robot manipulators is to generate a motion trajectory to move from an initial configuration to reach an end-effector in a goal position, while satisfying desired constraints (e.g. avoiding obstacles, joint limits, etc.). Specifically, avoiding obstacles is the principal constraint for kinematically redundant manipulators, which have high degrees-of-freedom (dofs). Recently, sampling-based path planners including the Rapidly-exploring Random Tree (RRT) [1] have been successfully used to give a probabilistically complete solution for redundant manipulators. Most sampling-based planners work in the configuration space (or joint space) and experience difficulty as the number of dimensions of the configuration space grows.

This phenomenon becomes worse for hyper-redundant manipulators whose configuration space has much higher dimensions than common task spaces (e.g. three dimensions when a task is positioning an end-effector in a 3D workspace). Normally more than 10-dofs for rigid link manipulators are considered to be hyper-redundant. These extra redundant dofs give the manipulator greater dexterity and potential for maneuvering within tight environments. Therefore robots for disaster relief or medical surgery are representative applications for hyper-redundant manipulators [2]. One example of hyper-redundant manipulators is the 30-dofs robot by Chirikjian and Burdick [3]. Extra redundancy also occurs computational inefficiency to control robots. Many approaches have been proposed for analyzing and planning hyper-redundant manipulators [4], [5], [6], [7].

Recently, a concept of utilizing the task space has received a strong attention for boosting the performance of planners. Some of them use a task space distance function [8], exploit the task space as a goal bias [9], or perform direct task space exploration [10]. Specifically, directly exploring the task space can enable planners to effectively solve high dimensional problems by using a much lower dimensional task space.

Main contributions. We propose a novel sampling strategy to improve the performance of a trajectory planner that directly explores the task space for hyper-redundant manipulators. For our sampling method, we propose to use a concept of productive regions, which effectively guide a random tree towards a goal state. We then propose two sampling domains for each node in the random tree: a promising maximum reachable area (MRA) and a detour sampling domain. We construct an MRA of a node in the task space such that it approximates a region that a manipulator can reach from the node by using an employed local planner without any collisions. When the MRA of a node contains the goal state, we classify it as a promising node and use its MRA as its sampling domain. For other nodes that do not contain the goal state, we call them unpromising and additionally construct a detour sampling domain for detouring obstacles constraining the manipulator.

To show benefits of our method, we compared our method against the standard RRT and the Task-Space RRT (TS-RRT) [10] across various dofs manipulators, which have $\mathbb{R}^2$ task space. Our experimental results show that our planner gives a solution on complex scenes with many obstacles faster by a factor of up to 3.54 than TS-RRT, which is the state-of-the-art task space planner. This improvement is mainly caused by effectiveness and efficiency of our sampling bias technique that covers productive regions. In addition, we have applied our method to a planning problem of cabled mobile robots by approximating a cabled robot to a manipulator with high dofs. Experimental results show that our planner gives higher success ratio (2.3 times) and better performance (3.6 times) than TS-RRT in a complicated environment.

II. RELATED WORK

Sampling-based single query motion planning algorithms (e.g., RRT [1] and EST [11]) have been used to solve various motion planning problems including high-dimensional manipulation planning. For overcoming the high dimensionality

[1]Junghwan Lee and [2]Sung-eui Yoon are at Dept. of CS, KAIST, Daejeon, South Korea goolbee@gmail.com, sungeui@gmail.com

of redundant manipulators, many techniques have been proposed to improve the performance of a planner by utilizing the task space, which has much lower dimensions compared to the configuration space of a robot.

Yao *et al.* [12] proposed ATACE, which first generates end-effector paths in the task space, and then tries to track the paths in the configuration space by using an end-effector trajectory tracking planner [13] as a local planner. Bertram *et al.* [8] used a heuristic goal function that estimates a distance from a sample to goals, in order to bias sampling in the configuration space. JT-RRT [9], proposed by Weghe *et al.*, combined the Jacobian transpose method with the standard RRT exploring the configuration space. Especially this method used a task space goal bias that attempts to connect nodes in the configuration space and the goal end-effector position by a local planner based on the Jacobian transpose method. Diankov *et al.* [14] proposed BiSpace Planning, which explores both configuration and task spaces in a similar manner to the bidirectional RRT approach; two different trees are grown from initial and goal states and the planner periodically attempts to connect them.

While planners mentioned above used workspace goal positions for biasing the configuration exploration so that planners could find a solution effectively, techniques exploiting the task space as its main exploration space have been proposed [10], [15]. A concept of directly exploring the task space is very helpful especially for a hyper-redundant manipulator, which has many dofs, because the dimension of the task space remains the same, while the dimension of the configuration space increases. As a result, the task space exploration is scalable with respect to the dof of robots. In order to reconstruct samples of the configuration space from ones of the task space, Shkolnit *et al.* [10] uses the Jacobian Pseudoinverse method, which is widely used for the task space control [16]. Behnisch *et al.* [15] further exploits the concept of the direct task space exploration by utilizing the null space velocity for avoiding obstacles.

Many variants of a sampling-based algorithm have been proposed in order to improve the performance of motion planners. The common idea of these techniques is to bias the sampling distribution towards more important regions [17], [18], [19], [20], [21]. While a biasing technique in the configuration space has been widely researched, biasing the task space sampling distribution has not been studied well, especially for a redundant manipulator.

III. PLANNING ALGORITHM

In this section, we give a brief overview of our global planner and its local planner as a preliminary. We then discuss issues that arise when a task space distance function is used for the planner.

A. Global Planner

Our global planner is based on the Task-Space RRT (TS-RRT) [10]. TS-RRT is similar to the standard RRT, but performs its sampling procedure in the task space. TS-RRT iteratively grows a tree whose nodes consist of both task space and configuration space states. In each iteration, TS-RRT generates a random sample, x_r, in the task space and finds its nearest neighbor node from its random tree. The nearest neighbor node is associated with $[x_n, q_n]$, which encodes its task space and configuration space states, respectively. An action u is then computed by a local planner (See III-B) to connect the nearest neighbor node and the random sample in the task space. Finally, a new configuration space state, q_{new} is computed by taking the action u from q_n. If there is no collision in between q_{new} and q_n, then a new node $[x_{new}, q_{new}]$ is added to the random tree, solving the forward kinematics equation with q_{new} to get the corresponding task space state x_{new}. Its pseudocode is shown in Algorithm 1.

Algorithm 1 Task-Space RRT, TS-RRT

Require: tree T
  T.AddVertex (x_{init}, q_{init})
  repeat
    $x_r \leftarrow$ RandomStateInTaskSpace()
    $[x_n, q_n] \leftarrow$ NearestNeighbor(x_r, T)
    $u \leftarrow$ Control($[x_n, q_n]$, x_r)
    $q_{new} \leftarrow$ NewState(q_n, u)
    if CollisionFree(q_{new}, q_n) **then**
      $x_{new} \leftarrow$ ForwardKinematics(q_{new})
      T.AddVertex($[x_{new}, q_{new}]$)
      T.AddEdge($[x_n, q_n]$, $[x_{new}, q_{new}]$)
    end if
  until a collision-free path between x_{init} and x_{goal} is found

B. Local Planner

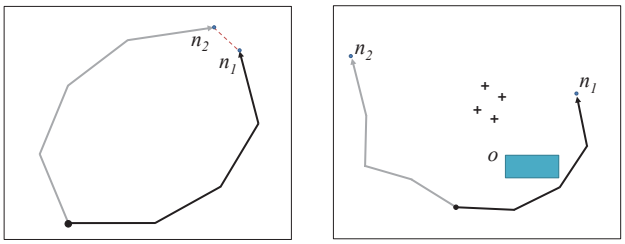
In sampling-based algorithms, a local planner is responsible for connecting two nodes in a sampling space. Since sampling is performed on the low dimensional task space in the global planner, the local planner should generate full trajectories in the configuration space following their task space trajectories. A naive way for generating such trajectories in the configuration space is to generate a random action until a manipulator reaches to its local goal. Unfortunately, this is very inefficient, especially for hyper-redundant manipulators whose configuration space has quite high dimensions. A better and well established approach is to exploit a feedback controller with inverse kinematics solutions [10].

Among many models to solve the inverse kinematics problem, the Jacobian pseudoinverse method is widely used in the control literature [22, pp. 245–268]. The Jacobian pseudoinverse solution for a configuration space velocity vector $\dot{q}$ can be written as:

$$\dot{q} = J^{\dagger}\dot{t} + (I - J^{\dagger}J)\dot{q}_0,$$

where $\dot{t}$ is a task space velocity vector representing the desired movement of an end-effector. Also, J and $J^{\dagger}$ represent the Jacobian and the Jacobian pseudoinverse, respectively.

The second term $(I - J^{\dagger}J)$ represents an orthogonal projection matrix in the null-space of J and $\dot{q}_0$ is an arbitrary configuration space velocity; therefore different configuration space velocities can be obtained based on a null-space



(a) (b)

Fig. 1. (a) shows two different configurations of the manipulator, whose root is represented as the black dot and end-effector as an arrow. Their distance in the configuration space is much farther than the distance in the task space (represented as the dotted line) defined as the distance of end-effector position. (b) shows an inefficiency of the task space distance function to the sampling-based planner. The planner chooses n_1 as the nearest neighbor to random samples (plus marks), not n_2, while extensions from n_1 to these random samples are unlikely to succeed because of a nearby obstacle o shown as the square box.

velocity, while fixing the same task space velocity. The null-space velocity is used for solving redundancy resolution via local optimization of additional motion objectives by defining a proper criterion such as joint limit avoidance, workspace centering [23], collision avoidance with obstacles [24], or a combination of multiple criteria [15].

C. Motivation

TS-RRT directly grows its random tree in the task space with a distance function defined also in the task space (e.g., L_2 distance function when the task space is defined for a position of an end-effector). It is a common practice to use the distance function defined in its sampling space, the task space for the TS-RRT.

This simple approach, however, does not consider behaviors of local planners considering the inverse kinematics, as we discussed in Sec. III-B. As a result, a task space distance function frequently leads to unsuccessful local planning, because a pair of nodes has a close distance in the task space, but a far distance in the configuration space; see its example shown in Fig. 1-(a). Such unsuccessful local planning causes inefficient performance, since its operations include inverse kinematics computation and several collision detection calls.

Fig. 1-(b) shows an example, where a planner tries to connect random samples shown in plus marks to a node n_1, which is selected as the nearest neighbor to these samples by the L_2 distance function defined on the task space. Extension trials from n_1 to these random samples (i.e., plus marks) tend to fail, because a nearby obstacle o restricts the movement of a manipulator. On the other hand, the connection would very likely succeed if extensions are tried from n_2 to those plus marks. This example shows that the task space distance function can be ineffective, since it does not consider behaviors of the local planner and obstacles.

The planner can be stuck into local minima, where further extensions do not lead to reach the goal, unless extensions from the tree explore around obstacles constraining the manipulator. Especially in environments with many obstacles, local minima arise very frequently and result in many unsuccessful local planning operations and collision detection checks. This problem becomes more pronounced with hyper-redundant manipulators, because of its high dimensionality.

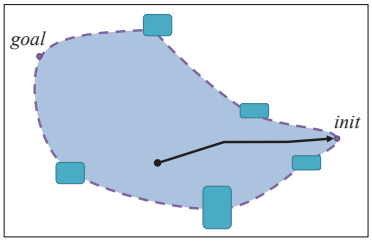


Fig. 2. An example of an ideal productive region given init and goal states. Boxes represent obstacles and the black dot represents the root of the manipulator. Sampling only in shaded regions is sufficient and efficient to move an end-effector (the black arrow) from init to goal.

IV. Sampling Bias to Productive Regions

In this section we explain our sampling bias method for productive regions towards a goal state. We first introduce the notion of productive regions and utilize a maximum reachable area (MRA) to identify productive regions. Specifically we propose two sampling domains: a promising MRA and a detour sampling domain, the union of which defines our productive regions. We then give an overview of our algorithm, followed by a detail of how to compute the detour sampling domain and the MRA.

A. Productive Regions-Oriented Sampling Heuristic

In order to efficiently reach to the goal state, we define *productive regions*, a set of task space states that have a high probability of leading the random tree to the goal state. Fig. 2 shows a conceptual example of the productive regions given initial and goal states. We then bias our sampling distribution such that random samples for exploration are more frequently generated on these regions, instead of sampling the whole task space uniformly.

Correctly identifying those regions itself, unfortunately, is a very hard problem, and its complexity is the same to that of the motion planning problem, because it requires full analysis of the whole environment and the movement of the manipulator in the high-dimensional configuration space. Instead, we approximately identify such productive regions considering only local geometry.

In order to bias our sampling on productive regions, we use a concept of MRA for each node in the random tree. An MRA of a node is defined as a group of task space states where a manipulator can reach from the node by an employed local planner without having collisions. The MRA serves as an upper bound of a region in the task space where the node can be extended to, given a local planner and nearby obstacles around the node.

Fig. 3 shows a conceptual example of an MRA. Gray lines represent examples of manipulator configurations that can be extended from a configuration n by the local planner without collisions. The node n cannot be directly extended to g, because a obstacle o constrains the movement of the manipulator. The planner should generate a path by detouring

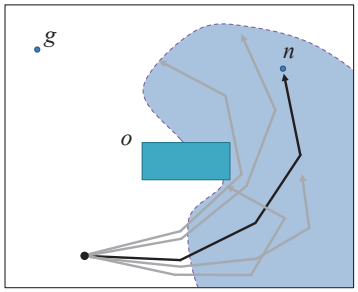


Fig. 3. A node n represents the end-effector of the manipulator (shown as line segments). Gray lines represent positions that the manipulator can reach by a local planner. On the other hand, the node n cannot reach to another node g by running a simple local planner because of a nearby obstacle o.

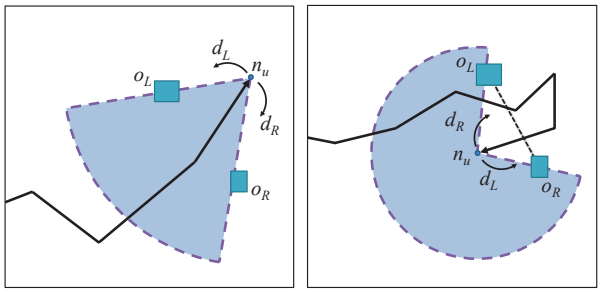
Algorithm 2 PROT-RRT

Require: tree T, q_{init}, x_{goal}
  $x_{init} \leftarrow$ ForwardKinematics(q_{init})
  $[x_{init}, q_{init}].MRA \leftarrow$ ComputeMRA()
  T.AddVertex ($[x_{init}, q_{init}]$)
  repeat
    repeat
      $x_r \leftarrow$ RandomStateInTaskSpace()
      $[x_n, q_n] \leftarrow$ NearestNeighbor(x_r, T)
    until inProductiveRegion(x_r, $[x_n, q_n]$)
    $u \leftarrow$ Control($[x_n, q_n]$, x_r)
    $q_{new} \leftarrow$ NewState(q_n, u)
    if CollisionFree(q_{new}, q_n) **then**
      $x_{new} \leftarrow$ ForwardKinematics(q_{new})
      $[x_{new}, q_{new}].MRA \leftarrow$ ComputeMRA()
      T.AddVertex($[x_{new}, q_{new}]$)
      T.AddEdge($[x_n, q_n]$, $[x_{new}, q_{new}]$)
    end if
  until a collision-free path between x_{init} and x_{goal} is found

the obstacle o in order to reach g. Fig. 6 shows a diagram on how to construct an MRA for the node n. We will explain how to compute the MRA in Sec. IV-C. In this section we give our overall algorithm utilizing MRAs.

We classify the MRA of a node to be *promising* or *unpromising*. We call the MRA promising, when the goal state is included in the area. Other MRAs are called unpromising. When the MRA of a node contains the goal state, exploration in that area is clearly productive for reaching the goal, since we can easily reach the goal state by using the local planner with those samples generated for exploration. As a result, the union of all the promising MRAs are a subset of approximated, productive regions; detour sampling domains are another subset of approximated, productive regions.

The random tree, however, tends not to have any promising MRAs during iterations, especially in its early stage, when the exploration has not been spread much. Sampling an unpromising MRA of a node is unlikely to lead the random tree to reach the goal state. Consequently, it is more desirable to bias our sampling in a way that we detour from the unpromising MRA constrained by obstacles, and extend towards other potentially promising regions. We therefore



(a) (b)

Fig. 4. (a) This figure shows a detour sampling domain for an unpromising node n_u in the $\mathbb{R}^2$ task space. (b) A detour sampling domain in $\mathbb{R}^2$ task space used for preventing to enter the local minima. In this case the manipulator passes multiple times through the virtual line (shown in the dotted black line) between two obstacles.

propose to use a simple detour sampling domain for an unpromising node, in order to detour obstacles constraining the manipulator. Fig. 4 shows an example of a detour sampling domain for an unpromising node n_u.

In order to generate samples in approximated productive regions, we first generate a sample from the task space in a uniform manner, and confine it to our productive regions, which are approximated by the union of promising MRAs and detour sampling domains. The resulting distribution becomes uniform in the approximated productive regions. The pseudocode of our algorithm is shown in Algorithm 2.

By exploring promising MRAs and detour sampling domains for unpromising nodes that approximate productive regions, we efficiently reach the goal state even for scenes with complex configurations of obstacles.

B. Computing Detour Sampling Domains

For an unpromising node, we first identify principal movement directions of an end-effector based on instantaneous directions computed by changing its joint values. In $\mathbb{R}^2$ space, principal movement directions of an end-effector are turning the end-effector into either its left or right directions, denoted as d_L and d_R, respectively (Fig. 4-(a)). We then compute an obstacle most constraining the manipulator in each movement direction. Fig. 4-(a) shows an example of our detour sampling domain of an unpromising node n in $\mathbb{R}^2$ task space. Constraining obstacles in directions of d_L and d_R are noted as o_L and o_R, respectively.

Computation of constraining obstacles to a certain direction can be implemented in many different ways. For example, we can compute the nearest obstacle overlapped with the span of the manipulator's movement to the given direction (d_L or d_R). Exact but expensive computation, fortunately, is unnecessary, mainly because we probabilistically bias our sampling based on computed obstacles. Among many alternatives, we chose to identify them by a simple method considering local geometry only. For each movement direction, we identify the nearest neighbor from the manipulator along the direction (e.g., d_L or d_R). When the nearest neighbor is located within a certain distance d from the end-link of the manipulator, we define it as the constraining

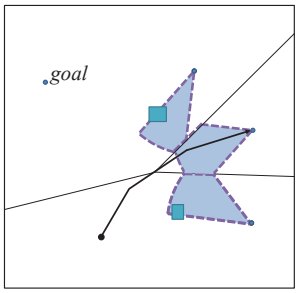


Fig. 5. In this case the manipulator should detour two obstacles (boxes) to reach the goal state. Shaded regions indicate detour sampling domains of nodes (blue dots) clipped by their Voronoi regions (black lines).

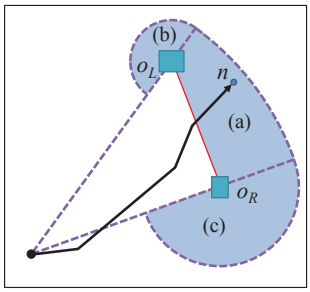


Fig. 6. Approximately constructed Maximum Reachable Area (MRA)

obstacle in that direction.

We assume that obstacles farther than d do not constrain the local movement of the manipulator. When obstacles are not found in distance d for a node, we use uniform sampling around the node instead. The distance d can be chosen depending on environments and manipulator's configurations. We set d to be 40% of the maximum distance of any two points in the task space. Fig. 5 shows an example of detour sampling domains for a set of unpromising nodes on regions constrained by nearby obstacles. Note that actual detour sampling domains are also clipped by the Voronoi regions of nodes, since each detour sampling domain is used when its node is chosen as the nearest node to a random sample.

An underlying assumption made so far for the detour sampling domain is that the manipulator passes only a single time through a virtual line drawn between two constraining obstacles o_L and o_R. There is, however, a case when the end-effector passes through the virtual line multiple times as shown in Fig. 4-(b). This kind of complication arises mainly because of the redundancy of the manipulator. In this case the sampling domain should be generated in an opposite direction, to prevent the manipulator re-entering the virtual line between two constraining obstacles. It can be distinguished simply by counting the number of links that pass through the virtual line between the constraining obstacles. When the number is even then the detour sampling domain is generated in an opposite direction.

C. Approximating the MRA

The exact computation of an MRA for each node is also challenging, because it requires to predict the movement of the high-dimensional manipulator by the local planner. Furthermore considering all those possible movements is an expensive computational operation. To reduce its complexity, we propose a simple approximation method for constructing the MRA. Our approximation considers only local geometry around the manipulator, instead of using global geometry information.

We identify obstacles that constrain the movement of the manipulator's end-effector. Specifically, we identify the nearest obstacles from a node in the same manner for computing constraining obstacles used for defining the detour sampling domain. Fig. 6 shows a schematic view on how to approximate the MRA for a node n. o_L and o_R represent two identified obstacles and the red line represents a base line for detouring in $\mathbb{R}^2$ task space. Moving the end-effector below the line is regarded as detouring from a region constrained by those two obstacles, and thus the regions below the line is not included in the MRA.

The MRA is approximately constructed by one circular sector (a), and two half-circles (b) and (c), shown in Fig. 6. The circular sector (a) represents an area where the end-effector can reach by stretching its joints. We thus compute the circular sector with its center located at the root of the manipulator and its radius is the length of the manipulator. Each side of the circular sector is computed by either one of two obstacles. One left half-circle (b) represents an area where the manipulator can reach with a physical contact between the left obstacle o_L and the manipulator. Its radius is set to the manipulator's length minus the distance between the root and the obstacle. In a similar manner the right half-circle (c) is computed with the right obstacle o_R.

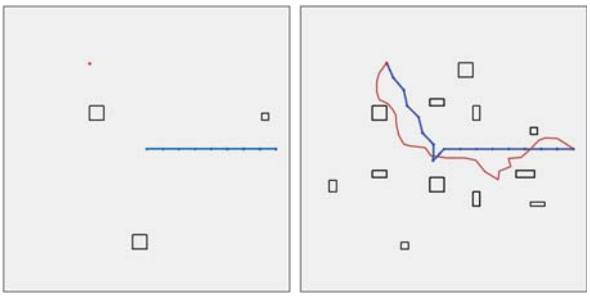
The overhead of computing approximated MRA is relatively very small, because the approximated MRA for each node is constructed with only few obstacles and simple geometric operations. In addition, this computation is performed only once for each node, when a node is newly added to the random tree.

D. Hybrid distribution

Constructed MRA and detour sampling domains for our productive regions are approximated by using only local geometry. This inaccuracy can aggravate the completeness of sampling-based planners. In order to avoid this issue and guarantee the probabilistic completeness for our method, we set our planner to enable our sampling bias with probability α and uniform sampling with probability $1-\alpha$. We set α to 0.8 in our experiments.

V. RESULTS AND DISCUSSIONS

We implemented our method, PROT, and performed various experiments on an Intel i7 desktop machine that has 3.6GHz CPU. Additionally, we compared PROT to the standard RRT and TS-RRT against a planning problem for a serial rigid link manipulator, which has n number of revolute joints. A given initial pose is defined in the configuration space as a rest pose, and a goal end-effector position is defined in $\mathbb{R}^2$ task space. Any goal configurations are acceptable, when the end-effector locates at the goal position



(a) Easy benchmark (b) Difficult benchmark

Fig. 7. Tested environments with a 8-dofs manipulator. The initial pose is shown in blue, and the goal state is shown in red. (a) An easy benchmark contains three obstacles (black rectangles) which do not interfere with the movement of the manipulator to reach goal state. (b) This benchmark contains many obstacles (black rectangles) and gives rise to many potential local minima. A solution trajectory of the end-effector is shown in red.

in the task space. For simplicity, we assume that the problem does not involve dynamics, and joints are unconstrained.

A. Results and Comparisons

We tested PROT against two different benchmarks: one has a few number of obstacles and thus is relatively easy to solve, and another has complex configurations of obstacles that lead to local minima regions, which make the problem difficult. We ran each test 100 times and report the mean of those results. For our experiments, we perform collision detection in a brute-force way checking all obstacles with every link of the manipulator. We can also use a more advanced and efficient collision detection technique [25]. Nonetheless our current collision detection unit takes a negligible cost.

Easy benchmark. We tested 8- and 20-dofs manipulators in an easy environment (Fig. 7-(a)) with three obstacles that do not interfere with the movement of the manipulator to reach goal state (red dot). The standard RRT takes much less time per node (about 0.94ms) than PROT (about 4.51ms) for the 20-dofs manipulator. The standard RRT quickly spreads the random tree in the configuration space, because of its cheap cost of generating a node. On the other hand, the local planner of PROT consisting of the Jacobian pseudoinverse and null-space calculations is expensive, leading to a higher cost for each node. Nonetheless, for the 8-dofs manipulator, the standard RRT does not give a solution within 60 s, while PROT finds a solution in 0.818 s on average. This is mainly because the sampling process of PROT is performed in the task space, more efficiently leading to reach the goal state. The performance improvements go higher as the dimensionality of the configuration space goes up. For the 20-dofs manipulator, PROT takes 2.436 s on average, while the standard RRT fails to solve the problem in 10 minutes. In the literature of motion planning, it is well known that the complexity of planning grows exponentially as the dimension grows. On the other hand, PROT avoids this problem by performing the sampling process in the task space, which is in $\mathbb{R}^2$ for the tested benchmark.

Compared with TS-RRT that also performs the sampling process in the task space, our planner does not show considerable improvements in this simple benchmark. This is mainly because obstacles do not cause any local minimum cases or constrain the manipulator's movement. As a result, the union of productive regions of our method are almost same to the whole task space. Note that this is the worst case that emphasizes the overhead of our method. The overhead includes computing MRAs and performing bias sampling, but it is small since it deals with only local geometry. Overall, our method showed a minor degradation, 12%, over TS-RRT for the 8-dofs manipulator and showed a slight improvement, 3% for the 20-dofs manipulator (Table I). These results acquired from the simple benchmark demonstrate the low overhead of our method even compared to TS-RRT.

TABLE I

EXPERIMENT RESULTS FOR THE SIMPLE BENCHMARK

8-dofs	RRT	TS-RRT	PROT
# of iterations	-	571.3	603.5
# of nodes	>53698	491.6	507.6
time (s)	>60	0.723	0.818
20-dofs	RRT	TS-RRT	PROT
# of iterations	-	708.7	666.8
# of nodes	>638295	510.9	540.1
time (s)	>600	2.519	2.436

TABLE II

EXPERIMENT RESULTS FOR THE DIFFICULT BENCHMARK

8-dofs	RRT	TS-RRT	PROT
# of iterations	-	1342.2	529.9
# of nodes	>789120	661.9	401.7
time (s)	>300	13.43	7.19
20-dofs	RRT	TS-RRT	PROT
# of iterations	-	1832.0	723.9
# of nodes	-	690.4	417.6
time (s)	-	66.94	18.89

Difficult benchmark. We also compared planners on a more challenging benchmark (Fig. 7-(b)) , which has many obstacles and gives rise to many potential local optima. The standard RRT was unable to compute a solution in 300 s and we did not test 20-dofs manipulator for the standard RRT. In this difficult benchmark, productive regions are much smaller than the whole task space, because of many obstacles in the environment. For 8-dofs manipulator, PROT shows 1.87 times improvement over TS-RRT with a uniform distribution (Table II). This result demonstrates that our planner with biased sampling can explore the task space more efficiently and find a solution much faster than the planner with the uniform distribution. PROT shows 3.54 times improvement for the 20-dofs manipulator. The improvement increases as we have a higher dimensional space, because the computation for one iteration consisting of local planning and several collision detections, takes higher portions of the overall computation time with a higher dimensional space.

Discussion. According to our experimentation, the computation of the Jacobian pseudoinverse local planner occupies

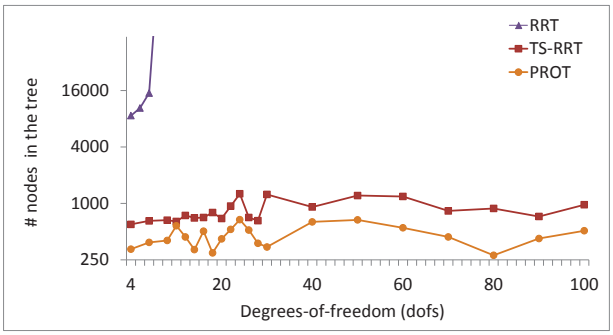


Fig. 8. The graph shows the total number of generated nodes in the tree after a first solution is found. Values in the Y-axis is on a log scale.

a large portion (about 70%) of the total running time, while the portion of collision detections is relatively small. The main benefit of our biased sampling distribution is to reduce the number of expensive local planning trials for expanding the random tree.

Our method shows its strength over TS-RRT, especially for handling complicated environments, where many obstacles exist such that the movement of the manipulator is highly restricted by those obstacles and thus the random tree is likely to be stuck in local minima regions. In Table II, the numbers of iterations and nodes represent to the total number of trials and their successes of local planning, respectively. TS-RRT generates more nodes in the random tree than our planner to solve the problem, which implies that the TS-RRT spends more time for generating ineffective nodes (e.g. in local minimum) to find a solution.

Scalability. PROT can efficiently compute a solution for high dofs manipulators, even for more than fifty dimensions. This is because the dimension of the task space where the sampling process is performed is independent from the dimensionality of manipulators. Fig. 8 shows a graph showing the number of nodes of the random tree generated after finding a solution, as a function of dimensions of manipulators with different methods. This experiment was conducted in the difficult benchmark. Although the number of nodes required for finding a solution grows super-linearly for the standard RRT, our method uses a much smaller number of nodes, which are within an almost fixed range across different dofs tested up to a 100-dofs manipulator. Overall TS-RRT has a similar tendency but the total number of nodes is larger. Furthermore, our method runs faster than TS-RRT across all of tested dofs. This result is attributed to our productive sampling regions that effectively avoid the local minima. The scalability of our method could lend itself to other problems such as high dimensional snake-arm robots or continuum manipulators that might be used for diverse applications [26].

B. Application to Cabled Mobile Robots

We have applied our method to a cabled mobile robot problem whose goal is to find a trajectory of a mobile robot that is connected to a fixed point by an under-actuated constant length cable. This problem is difficult, since the

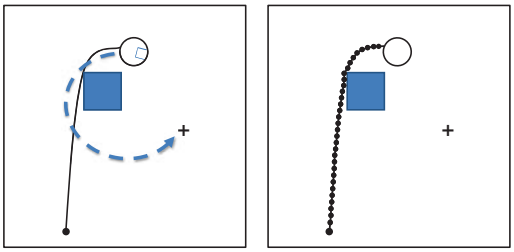


Fig. 9. (left) An example where the cabled mobile robot should detour the obstacle (squared box) to reach to the goal position (the plus mark). (right) An approximated 33-dofs manipulator from the cabled mobile robot. The end-effector is represented by the same-sized circle as the mobile robot, and dots represent to joints of the manipulator.

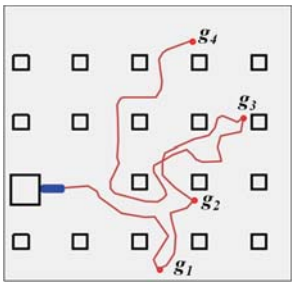


Fig. 10. This figure shows a test environment for a cabled mobile robot. The robot should reach goal positions (red dots) sequentially by avoiding obstacles (squared boxes). The start position is located near the large box and the robot's cable (blue lines) is approximated by a 100-dofs manipulator. The cable is piled initially. The computed solution trajectory is shown in the red curve. The end-effector is moved back to reach g_4 from g_3.

cable imposes constraints on the movement of the mobile robot: the cable can collide with obstacles and make the robot moving no further than the cable length. Fig. 9 shows an example where the cabled robot should detour obstacles constraining the cable to reach the goal position, because the direct movement to the goal is restricted by the contacts between the cable and obstacles. The motion planning problem that the cabled mobile robot faces is essentially same to that of our redundant manipulators. The difference is that the cable cannot be actuated directly but follows the mobile robot's movement by respecting only the kinematics of links representing the cable.

We approximate the cabled robot problem by the manipulator planning problem. Specifically we treat a cabled robot as a redundant manipulator that has an infinite number of dimensions. In practice, we have found that one hundred dimensions are enough to approximate the movement of the real cable. The end-effector of our manipulator then represents the position of the mobile robot. In other words, the trajectory of the manipulator's end-effector acts as a feasible path for the mobile robot that guides the cable to move around obstacles. We let the cabled mobile robot to follow the solution trajectory of the manipulator's end-effector.

Fig. 10 and Table III show a test environment used for the cabled mobile robot and its results, respectively. In the test the robot starts from its base and visits four goal positions sequentially. We approximate the cabled robot with a 100-dofs manipulator. We compared our method and TS-RRT; we ran

TABLE III

EXPERIMENT RESULTS FOR THE CABLED MOBILE ROBOT BENCHMARK

	TS-RRT	PROT
# of iterations	6563.9	2752.8
# of nodes	1300.5	527.3
time (s)	250.46	69.24
Success ratio	35%	80%

100 times for each algorithm with a maximum running time of 800 seconds. PROT shows 3.61 times faster performance than TS-RRT. We also measure a success ratio of finding a solution given the maximum running time out of 100 tests. Our planner's success ratio, 80%, is much higher than that, 35%, of TS-RRT. These results demonstrate advantages of our method over TS-RRT in the tested application.

VI. CONCLUSION

We have presented a novel sampling strategy, Productive Regions-Oriented Task space (PROT), to improve the performance of a task space sampling-based planner for hyper-redundant manipulators. Our planner directly explores the task space with a biased sampling distribution for effectively exploring productive regions that lead a random tree towards a goal state. In order to approximate the productive regions, we proposed two sampling domains: a maximum reachable area (MRA) and a detour sampling domain. The union of these domains defines our productive regions. We approximately and efficiently identify constraining obstacles for computing MRAs and detour sampling domains by considering only local geometry around the manipulator instead of using global geometry information. Our experimental results show that our sampling heuristic is effective to solve the problem compared to the previous task space planner [10]. Our planner shows up to 3.54 times improvements for a 20-dofs manipulator in a challenging benchmark. We have also applied our method to solve a cabled mobile robot problem by approximating the problem as a 100-dofs manipulator planning problem.

There are many interesting future research directions. We would like to compute productive regions in various task spaces. Currently, we have experimented $\mathbb{R}^2$ task space and thus the manipulator has only two principal movement directions. In order to extend the methods to $\mathbb{R}^3$ task space, a combinatorial analysis of the manipulator configuration is needed to identify its principle movement directions for the computation of the MRA and the detour sampling domain. We also would like to extend our method for various manipulators with prismatic joints or multiple end-effectors. Finally, we would like to combine asymptotically-optimal algorithms [27] within our method for finding the optimal solution.

ACKNOWLEDGMENTS

We would like to thank SGLab members and anonymous reviewers for constructive comments. This work was supported in part by NRF-2013R1A1A2058052, DAPA/ADD (UD110006MD), MEST/NRF (2013-067321), and IT R&D program of MOTIE/KEIT [10044970]. Sung-eui Yoon is a corresponding author of the paper.

REFERENCES

[1] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *ICRA*, 2000, pp. 995–1001.

[2] K. Ikuta, M. Nokata, and S. Aritomi, "Biomedical micro robots driven by miniature cybernetic actuator," in *Proc. of IEEE Workshop on Micro Electro Mechanical Systems*. IEEE, 1994, pp. 263–268.

[3] G. S. Chirikjian and J. W. Burdick, "Design and experiments with a 30 dof robot," in *ICRA*, 1993, pp. 113–119.

[4] ——, "An obstacle avoidance algorithm for hyper-redundant manipulators," in *ICRA*. IEEE, 1990, pp. 625–631.

[5] I. Ebert-Uphoff and G. S. Chirikjian, "Inverse kinematics of discretely actuated hyper-redundant manipulators using workspace densities," in *ICRA*, vol. 1. IEEE, 1996, pp. 139–145.

[6] S. Ma and M. Konno, "An obstacle avoidance scheme for hyper-redundant manipulators-global motion planning in posture space," in *ICRA*. IEEE, 1997, pp. 161–166.

[7] B. Dasgupta, A. Gupta, and E. Singla, "A variational approach to path planning for hyper-redundant manipulators," *Robotics and Autonomous Systems*, vol. 57, no. 2, pp. 194–201, 2009.

[8] D. Bertram, J. Kuffner, R. Dillmann, and T. Asfour, "An integrated approach to inverse kinematics and path planning for redundant manipulators," in *ICRA*, 2006, pp. 1874–1879.

[9] M. Vande Weghe, D. Ferguson, and S. S. Srinivasa, "Randomized path planning for redundant manipulators without inverse kinematics," in *7th IEEE-RAS Int. Conf. on Humanoid Robots*, 2007, pp. 477–482.

[10] A. Shkolnik and R. Tedrake, "Path planning in 1000+ dimensions using a task-space voronoi bias," in *ICRA*, 2009, pp. 2061–2067.

[11] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Int. Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

[12] Z. Yao and K. Gupta, "Path planning with general end-effector constraints," *Robotics and Autonomous Systems*, vol. 55, no. 4, pp. 316–327, 2007.

[13] A. A. Maciejewski and C. A. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments," *Int. Journal of Robotics Research*, vol. 4, no. 3, pp. 109–117, 1985.

[14] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa, and J. Kuffner, "Bispace planning: Concurrent multi-space exploration," *Proceedings of Robotics: Science and Systems IV*, 2008.

[15] M. Behnisch, R. Haschke, and M. Gienger, "Task space motion planning using reactive control," in *IROS*, 2010, pp. 5934–5940.

[16] A. Liegeois, "Automatic supervisory control of the configuration and behavior of multibody mechanisms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 7, no. 12, pp. 868–871, 1977.

[17] A. Yershova, L. Jaillet, T. Simeon, and S. LaValle, "Dynamic-domain rrts: Efficient exploration by controlling the sampling domain," in *ICRA*, 2005, pp. 3856 – 3861.

[18] S. Rodriguez, X. Tang, J.-M. Lien, and N. M. Amato, "An obstacle-based rapidly-exploring random tree," in *ICRA*, 2006, pp. 895–900.

[19] L. Zhang and D. Manocha, "An efficient retraction-based rrt planner," in *ICRA*, 2008, pp. 3743 –3750.

[20] S. Dalibard and J. Laumond, "Linear dimensionality reduction in random motion planning," *Int. Journal of Robotics Research*, 2011.

[21] J. Lee, O. Kwon, L. Zhang, and S. Yoon, "Sr-rrt: Selective retraction-based rrt planner," in *ICRA*, 2012, pp. 2543–2550.

[22] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer, 2008.

[23] L. Sentis and O. Khatib, "Control of free-floating humanoid robots through task prioritization," in *ICRA*, 2005, pp. 1718–1723.

[24] O. Khatib, "Real-time obstable avoidance for manipulators and mobile robots," *Int. Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[25] D. Kim, J.-P. Heo, J. Huh, J. Kim, and S.-E. Yoon, "HPCCD: Hybrid parallel continuous collision detection," *Comput. Graph. Forum (Pacific Graphics)*, vol. 28, no. 7, 2009.

[26] R. Buckingham, "Snake arm robots," *Industrial Robot: An International Journal*, vol. 29, no. 3, pp. 242–245, 2002.

[27] A. Perez, S. Karaman, A. Shkolnik, E. Frazzoli, S. Teller, and M. R. Walter, "Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms," in *IROS*, 2011, pp. 4307–4313.