Addressing Cost-Space Chasms in Manipulation Planning

Dmitry Berenson[1] Thierry Siméon[2] Siddhartha S. Srinivasa[1,3]

Abstract*— Finding paths in high-dimensional spaces becomes difficult when we wish to optimize the cost of a path in addition to obeying feasibility constraints. Recently the T-RRT algorithm was presented as a method to plan in high-dimensional cost-spaces and it was shown to perform well across a variety of problems. However, since the T-RRT relies solely on sampling to explore the space, it has difficulty navigating *cost-space chasms*–narrow low-cost regions surrounded by increasing cost. Such chasms are particularly common in planning for manipulators because many useful cost functions induce narrow or lower-dimensional low-cost areas. This paper presents the GradienT-RRT algorithm, which combines the T-RRT with a local gradient method to bias the search toward lower-cost regions. GradienT-RRT is effective at navigating chasms because it explores low-cost regions that are too narrow to explore by sampling alone. We compare the performance of T-RRT and GradienT-RRT on planning problems involving cost functions defined in workspace, task space, and C-space. We find that GradienT-RRT outperforms T-RRT in terms of the cost of the final path while maintaining better or comparable computation time. We also find that the cost of paths generated by GradienT-RRT is far less sensitive to changes in a key parameter, making it easier to tune the algorithm. Finally, we conclude with a demonstration of GradienT-RRT on a planning-with-uncertainty task on the physical HERB robot.

I. Introduction

Planning paths for manipulators becomes more difficult when we wish to optimize the cost of the path in addition to satisfying feasibility constraints. The high dimensionality of the problem precludes the exhaustive computation necessary to find the optimal path.

Recently the Transition-based RRT (T-RRT) [1] was presented as a method to manage the growth of a search tree in a high-dimensional cost-space. This algorithm uses a process similar to stochastic optimization methods, where a temperature parameter decides whether to accept a certain transition. When the T-RRT is stuck in a local minimum (i.e. many extensions are being rejected), the temperature is increased to allow more exploration. When the algorithm accepts a higher-cost extension, the temperature is decreased so the tree does not over-explore high-cost regions. In this way, T-RRT is biased to explore lower-cost regions before allowing higher-cost extensions which may be necessary to find a feasible path.

The T-RRT was shown to be very successful on many problems and consistently outperforms the standard RRT [2] and the heuristically-guided RRT [3]. However, we have

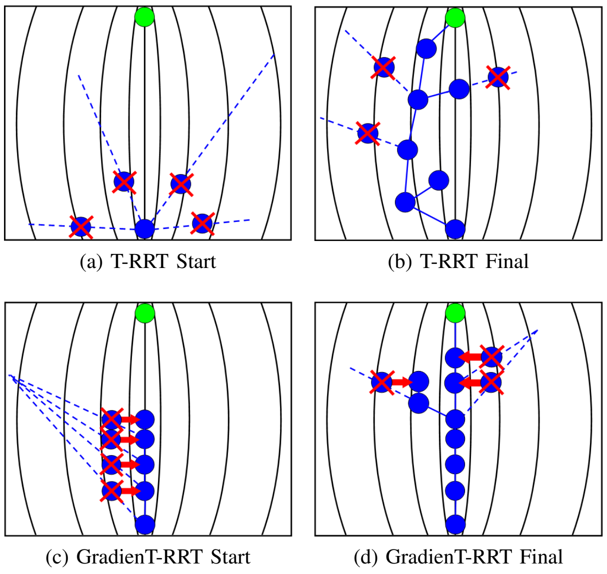

(a) T-RRT Start (b) T-RRT Final
(c) GradienT-RRT Start (d) GradienT-RRT Final

Fig. 1. Illustration of the two algorithms' performance in a cost-space chasm. The left image of each pair shows the state of the tree before a temperature increase and the right image shows the tree after the increase. The T-RRT is unable to sample the bottom of the chasm and increases temperature while GradienT-RRT uses gradient-descent (red arrows) to make progress before increasing temperature. GradienT-RRT generates a higher-quality path because it is able to generate nodes at the bottom of the chasm.

found that the T-RRT does not perform well for cost functions that induce *cost-space chasms*–narrow or even lower-dimensional low-cost passages around which cost increases. Such cost functions are especially relevant in manipulation planning where we may seek to penalize deviation from a lower-dimensional constraint on end-effector pose or navigate a narrow low-cost region in a manipulator's workspace. The T-RRT struggles in the cost-space chasms created by such cost functions because it relies on the sampling strategy of the RRT, which makes sampling in such lower-dimensional or narrow regions impossible or extremely unlikely. Thus almost every extension becomes a cost increase if the starting configuration is at the bottom of the chasm. As a result, the T-RRT explores the chasm slowly and the path generated lies on the higher-cost sides of the chasm, not on the lower-cost bottom (see Figure 1). Cost-space chasms thus significantly hinder the performance of the T-RRT for many types of cost functions relevant to manipulation planning.

To address this problem we present the GradienT-RRT algorithm, which combines the strengths of the T-RRT with local gradient-descent. The algorithm works by the same principles as the T-RRT, except that a gradient step is in-

[1]The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 15213, USA dberenso@cs.cmu.edu [2]CNRS ; LAAS and Université de Toulouse; UPS, INSA, INP, ISAE, LAAS; F31077 Toulouse, France nic@laas.fr [3]Intel Labs Pittsburgh, Pittsburgh, PA, 15213, USA siddhartha.srinivasa@intel.com

cluded during the extension. The gradient of the cost function allows this algorithm to explore spaces that are too narrow or lower-dimensional to explore by sampling alone. A key issue with incorporating the gradient is to avoid trapping the tree in local minima and to retain the beneficial exploration properties of the T-RRT.

In the follow sections we describe the T-RRT, formally define cost-space chasms, and show why they hinder the performance of the T-RRT. We then introduce the GradienT-RRT and discuss the trade-offs inherent in using the gradient. To show the versatility and practicality of our approach we also present general cost functions in workspace, task space, and C-space, and show how to compute their gradients efficiently. We then compare the performance of GradienT-RRT and T-RRT on three example problems where these cost functions induce cost-space chasms. Finally, we demonstrate a real-world implementation of GradienT-RRT, where it is used to plan a path amongst uncertain workspace occupancy for the HERB robot.

II. Related Work

The GradienT-RRT algorithm builds on both the T-RRT [1] and gradient methods often used in control. These methods seek to minimize a given function in the neighborhood of the robot's current configuration through gradient-descent. For instance, controllers have been developed for balancing legged robots [4], placing the end-effector somewhere in task space [5], and collision-avoidance [6].

Related planners that plan in the C-space while optimizing cost are the heuristically-guided RRT [3] and the Anytime RRT [7]. Though they perform well in mobile-robotics domains, these planners have difficulty in high-dimensional manipulation problems with continuous cost functions because an adequate heuristic is not readily available. Other planners that consider the *obstacleness* of a configuration [8] have difficulty with arbitrary cost functions because the cost threshold growth rate parameter is highly problem-dependent. Another related planner is Conformational Roadmaps [9], which uses a transition test similar to the T-RRT to explore molecular energy landscapes.

A common approach in motion planning is to attempt optimization only after a feasible path has been found. Methods like shortcut smoothing [10] or partial-shortcut [11] can be used to optimize the length and distance from obstacles of a feasible path. However, such methods only improve a given path locally. In many problems (like the ones presented in Section VII), shortcutting will not produce an adequately low-cost path from an arbitrary path because it is unlikely to find a cost-space chasm.

The problem of navigating cost-space chasms is related to that of exploring narrow passages in sampling-based planning. The integration of a gradient method and sampling-based planner in the GradienT-RRT is related to retraction-based methods [12][13] designed to explore narrow passages. Other methods of addressing narrow passages, such as bridge-sampling [14], diffusion control [15], and dilation-based approaches [16] are also widely studied in motion planning. Our problem domain differs because we consider continuous-valued cost functions, which are not narrow passages in terms of feasibility. It is unclear how to generalize the methods cited above to this cost-space domain. To our knowledge this is the first paper to address the cost-space chasm problem in the context of sampling-based planning.

III. T-RRT

The purpose of the T-RRT is to find feasible low-cost paths through high-dimensional cost spaces. The algorithm manages the trade-off between optimality and exploration by using a transition test similar to stochastic optimization methods. T-RRT maintains a temperature value $temp$ which determines the probability of allowing a higher-cost node to be added to the tree. $temp$ is automatically tuned by the algorithm and its behavior is controlled through several parameters, the most important being $nFailMax$. $nFailMax$ determines how many higher-cost nodes are rejected before increasing temperature.

We show a bi-directional implementation of the T-RRT in Algorithms 1 and 2. The algorithm is identical to the Bidirectional RRT except for the call to the ConsiderCost function (Algorithm 3 with GradienTRRT = False) in the extension. This function checks if a new configuration q_s has cost less than or equal to the cost of its parent. If it does, q_s is added to the search tree and the extension continues. If not, the cost difference is put through a temperature check. If it passes, the configuration is added to the tree, $temp$ is decreased by a factor of α, and $nFail$ is set to 0. If not, $nFail$ is incremented and the extension terminates. If $nFail$ reaches $nFailMax$, $temp$ is increased by a factor of α and $nFail$ is set to 0.

Thus the T-RRT tries growing lower-cost nodes $nFailMax$ number of times before allowing a higher probability of accepting a higher-cost node. Such an approach is quite reasonable because we want the algorithm to explore low-cost regions while avoiding being trapped in local minima. The $nFailMax$ parameter effectively controls the quality vs. search-time of the algorithm. Higher $nFailMax$ values bias the algorithm toward reducing cost at the expense of exploration. Thus the search takes more time for higher $nFailMax$ if a cost increase is necessary to find a feasible path.

IV. Cost-Space Chasms

Although the T-RRT works quite well in many scenarios, we found that it did not perform well when it needed to traverse narrow regions of low cost surrounded by regions of increasing cost. These structures are especially common in manipulation planning problems. For example, they arise from cost functions that bias the planner to imitate a demonstrated path, prefer an end-effector orientation, or avoid areas of uncertain occupancy in the workspace (see Section VII).

The cost functions in these problems create narrow or lower-dimensional low-cost regions surrounded by increasing cost. We call this type of structure a *cost-space chasm*, which we will define formally by introducing a property called c-*goodness* (inspired by ϵ-*goodness* [17]).

Let $\mathcal{Q}$ denote the C-space and $\mathcal{F} \subseteq \mathcal{Q}$ denote the free C-space. For a point $p \in \mathcal{F}$, let $S(p)$ consist of those points of $\mathcal{F}$ that can be connected to p by a straight line. For a $\mathcal{X} \subseteq \mathcal{Q}$, let $\mu(\mathcal{X})$ denote its volume.

If the cost of configurations were uniform (i.e. we consider only obstacles) a useful measure of the narrowness of the space at some point $p \in \mathcal{F}$ would be the ratio of the volume reachable from that point by a straight line to the volume defined by the sampling bounds $(\mu(S(p))/\mu(Q))$. However, if the cost of configurations in $S(p)$ is not uniform, it will be more difficult to reach some configurations in $S(p)$ than others using a planner like the T-RRT. The work along the line segment from p to $q \in S(p)$ determines the level of difficulty and it is defined as:

$$W(p,q) = \int_{\mathcal{L}_{pq}^{+}} \frac{\partial C(\mathcal{L}_{pq}(t))}{\partial t} dt \tag{1}$$

where $C(\cdot)$ is the cost of a configuration, $\mathcal{L}_{pq}$ is the line segment from p to q, and $\mathcal{L}_{pq}^{+}$ is the portion of the line segment where the cost function has positive slope. This function captures the amount of cost-increase we must overcome to move from p to q. Note that the definition of work in [1] contains another term that accounts for the length of $\mathcal{L}_{pq}$, but it is not relevant for computing c-goodness.

We would now like to compute the *work-weighted* volume of $S(p)$ and compare that to the volume of the C-space. We weigh all $q \in S(p)$ using a function $f(p,q)$ and the work-weighted volume of $S(p)$ is the integral of $f(p,q)$ over all $q \in S(p)$. $f(p,q)$ should have the following properties: $f(p,q) \in [0,1]$, $f(p,q_1) < f(p,q_2)$ if and only if $W(p,q_1) > W(p,q_2)$, and $f(p,q_1) = f(p,q_2)$ if and only if $W(p,q_1) = W(p,q_2)$. A weight of $f(p,q) = 1$ should mean that moving from p to q requires no work and $f(p,q) = 0$ should mean that it requires infinite work. There are many ways to define an $f(p,q)$ with these properties; for example $f(p,q) = e^{-W(p,q)/k}$ for some positive constant k.

If a p satisfies the equation

$$\int\limits_{S(p)} f(p,q) dq \geq c\mu(\mathcal{Q}) \tag{2}$$

for a non-negative c, then we say that p is c-good. Defining $f(p,q)$ according to the above requirements produces what we would expect for the uniform-cost case: $f(p,q) = 1$ for all q and the left side of Equation 2 becomes $\mu(S(p))$. Thus c-goodness for the uniform-cost case is simply a measure of the narrowness of the space $(\mu(S(p)) \geq c\mu(Q))$.

A cost-space chasm is then a set of connected configurations with low c-goodness. We term such sets "cost-space chasms" because they resemble the deep steep-sided chasms between mountains when visualized.

Despite the T-RRT's success in many scenarios, we found that it did not perform well for high-dimensional problems involving such chasms. To understand why, consider a node of the tree that is near the bottom of a chasm. This node

Algorithm 1: Bidirectional T-RRT(q_s, q_g)

```
$T_a$.Init($q_s$); $T_b$.Init($q_g$);
$T_a$.temp = $T_b$.temp = initTemp;
while TimeRemaining() do
    $q_{rand} \leftarrow$ RandomConfig();
    $q^a_{near} \leftarrow$ NearestNeighbor($T_a$, $q_{rand}$);
    $q^a_{reach} \leftarrow$ Extend($T_a$, $q^a_{near}$, $q_{rand}$);
    $q^b_{near} \leftarrow$ NearestNeighbor($T_b$, $q^a_{reach}$);
    $q^b_{reach} \leftarrow$ Extend($T_b$, $q^b_{near}$, $q^a_{reach}$);
    if $q^a_{reach}$ = $q^b_{reach}$ then
        $P \leftarrow$ ExtractPath($T_a$, $q^a_{reach}$, $T_b$, $q^b_{reach}$);
        return SmoothPath($P$);
    else
        Swap($T_a$, $T_b$);
    end
end
return NULL;
```

Algorithm 2: Extend(T, q_{near}, q_{target})

```
$q_s \leftarrow q_{near}$; $q^{old}_s \leftarrow q_{near}$;
while true do
    if $q_s = q_{target}$ then
        return $q_s$;
    else if $\|q_{target} - q_s\| > \|q^{old}_s - q_{target}\|$ then
        return $q^{old}_s$;
    end
    $q^{old}_s \leftarrow q_s$;
    $q_s \leftarrow q_s + \min(\Delta q_{step}, \|q_{target} - q_s\|) \frac{(q_{target} - q_s)}{\|q_{target} - q_s\|}$;
    $q_s \leftarrow$ ConsiderCost($T, q^{old}_s, q_s$);
    if $q_s \neq$ NULL then
        $T$.AddVertex($q_s$, $c$);
        $T$.AddEdge($q^{old}_s$, $q_s$);
    else
        return $q^{old}_s$;
    end
end
```

Algorithm 3: ConsiderCost(T, q^{old}_s, q_s)

```
if not CollisionFree($q^{old}_s$, $q_s$) then return NULL;
if $C(q_s) < C(q^{old}_s)$ then return $q_s$;
$d_{ij} \leftarrow \|q^{old}_s - q_s\|$;
$p \leftarrow exp\left(\frac{-(C(q_s) - C(q^{old}_s))/d_{ij}}{T.temp}\right)$;
if Rand$(0,1) < p$ then
    $T.temp \leftarrow T.temp/\alpha$;
    $T.nFail \leftarrow 0$;
    return $q_s$;
else
    if GradienTRRT then
        $\nabla q \leftarrow$ GetGradient($q_s$);
        $q_g \leftarrow q_s - \nabla q$;
        $d_{ij} \leftarrow \|q^{old}_s - q_g\|$;
        $p \leftarrow exp\left(\frac{-(C(q_g) - C(q^{old}_s))/d_{ij}}{T.temp}\right)$;
    end
    if $T.nFail > T.nFailMax$ then
        $T.temp \leftarrow \alpha(T.temp)$;
        $T.nFail \leftarrow 0$;
    else
        $T.nFail \leftarrow T.nFail + 1$;
    end
    if GradienTRRT and CollisionFree($q^{old}_s, q_g$) and
    ($C(q_g) < C(q^{old}_s)$ or Rand$(0,1) < p$) then
        return $q_g$;
    end
    return NULL;
end
```

will have low c-goodness, which means that we have a low probability of generating a collision-free extension that is not a significant cost increase. The T-RRT's exploration in this type of scenario is thus quite slow and the resulting paths lie on the higher-cost sides of the chasm, not on the bottom.

V. GradienT-RRT

To address these drawbacks, we propose combining the T-RRT with local gradient-descent of the cost function. Because gradient-descent does not rely on sampling to find lower-cost configurations, we can use it to navigate cost-space chasms. However, we must be very careful in how we include the gradient. For example, a naive strategy would be to apply the gradient at every node generated by the T-RRT. Doing so would inhibit the T-RRT's exploration and cause the tree to be trapped in local minima.

The GradienT-RRT algorithm differs from T-RRT in the ConsiderCost function (Algorithm 3 with GradienTRRT = True). If the configuration q_s is rejected (because it is higher-cost and it failed the temperature check), then we compute the gradient at q_s and take a step from q_s in the direction of the gradient. This produces the configuration q_g. If q_g is lower cost than the parent of q_s or passes the temperature check, we accept it. If q_g has a higher cost and fails the temperature check, we reject it and terminate the extension.

It is important to note that the $nFail$ counter and the temperature are not affected by the acceptance/rejection of q_g in GradienT-RRT; $nFail$ is only modified with respect to the cost of q_s. Not modifying $nFail$ and the temperature for gradient nodes is important because sometimes nodes generated using the gradient are only slightly higher in cost than the parent of q_s (for instance consider a tree trying to grow out of a cost-space bowl; the gradient will push the extension back into the bowl, as in Figure 2a). Such nodes will pass the temperature check and decrease the temperature, which can trap the tree in a local minimum (effectively causing it to spiral around the bowl) and prevent the temperature from increasing significantly.

Incorporating the gradient in this way allows us to preserve the exploration properties of the T-RRT while addressing its weakness in navigating cost-space chasms. However, in situations where the gradient does not help the tree grow, GradienT-RRT may perform slower than T-RRT (see Figure 2), although it will likely find a lower-cost path.

GradienT-RRT also inherits the probabilistic completeness of T-RRT because it generates the same nodes as T-RRT plus extra nodes resulting from the gradient steps. Since generating extra nodes can only increase coverage of the feasible manifold as time goes to infinity, the probabilistic completeness of T-RRT is preserved.

VI. Costs and Gradients

The GradienT-RRT method relies on a fast computation of the gradient of the cost at a given configuration. For robot manipulators, many useful cost functions reside in one of three spaces: workspace, task space, and C-space. We present a general cost function for each space that is useful for manipulation planning and describe how to compute the cost of a configuration $C(q)$ as well as the gradient ∇q efficiently.

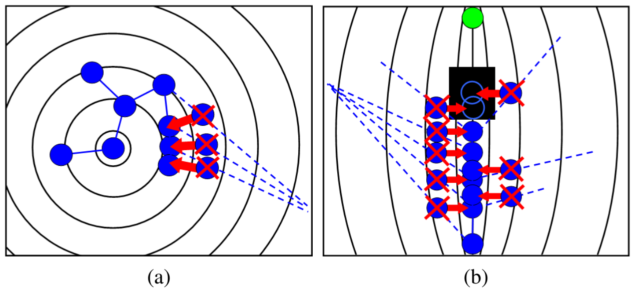


Fig. 2. Illustration of two difficult situations for GradienT-RRT. (a) The tree must grow out of a cost-space bowl. (b) The tree's progress in a chasm is blocked by an obstacle. In both of these situations, GradienT-RRT will generate more nodes in the low-cost region than T-RRT without making significant progress (the number of extra nodes depends on the $nFailMax$ parameter). This will slow down the algorithm, however the tree will eventual grow out of the bowl and around the obstacle as the temperature increases.

A. Workspace Costs

Workspace constraints are the most common constraints in motion planning, where they are used to represent obstacles. However these constraints are usually binary; either the robot is in collision or it is not. Imagine, however, that the robot was planning in a workspace with uncertain occupancy, so that the cost of a configuration depended on the belief of the robot being in collision. Our approach to computing the cost and gradient for such a scenario is similar to that used in CHOMP [18], which considers distance to obstacles as the cost of a configuration.

Let the workspace cost be $C_w(x)$ for $x \in \mathbb{R}^3$. $C(q)$ is then the integral of $C_w(x)$ over $x \in R(q)$, where $R(q)$ is the space occupied by the robot at the configuration q. Since computing the exact integral is prohibitively expensive, we perform the following process to quickly approximate its value.

First, we can pre-compute a voxelization of $C_w(x)$ to save on computation time when evaluating $C(q)$ in the planner. To do this, we discretize the workspace into a voxel grid V. Each voxel $v \in V$ is assigned a cost $C_w(v)$ from the workspace cost at its center, which reflects the desirability of occupying v with any part of the robot. In the uncertainty example above, $C_w(v)$ would depend on the probability of v being occupied by an obstacle (see Figure 3a).

We also pre-compute a set of points within the volume of each of the robot's links. When the robot is placed in a given configuration, we transform the points inside each link by the link's pose and check which voxels the points occupy. The sum of the voxel cost for each point is the approximated cost of the configuration. Note that this process can be made arbitrarily accurate by adding more points and by increasing the resolution of V, though the time needed to evaluate the cost will increase.

To obtain the points in the robot's volume, we first divide each robot link into simplices using a combination of convex decomposition [19] and Delaunay Triangulation. We then choose a simplex with probability proportional to its volume and generate a sample point within that simplex by taking a random convex combination of its vertices [20]. We repeat this process for some number of iterations to generate a uniformly sampled set of points P in the robot's volume. The sampling can also be biased toward lower-volume but more important parts of the robot (such as the fingers) by biasing the selection of simplices (see Figure 3b).

Besides evaluating the cost of a configuration, GradienT-RRT requires the gradient of the cost function. To compute the gradient ∇x of a point $x \in R(q)$, we can take the partial derivative of the cost function along each spatial dimension and evaluate it at x. However, ∇x exists in the workspace and we must transform this vector into the C-space in order to obtain ∇q. We can do this through the Jacobian $\mathbf{J}(q, x)$, which is defined by:

$$\dot{x} = \mathbf{J}(q, x)\dot{q} \tag{3}$$

We can then generalize the Jacobian to account for multiple points $x_1, x_2, ... \in R(q)$

$$\mathbf{J}(q, x_1, x_2, ...) = \begin{bmatrix} \mathbf{J}(q, x_1) \\ \mathbf{J}(q, x_2) \\ \vdots \end{bmatrix} \tag{4}$$

The C-space gradient is then

$$\nabla q = \mathbf{J}(q, x_1, x_2, ...)^T \left[\nabla x_1^T, \nabla x_2^T, ...\right]^T \tag{5}$$

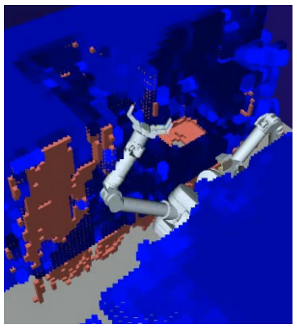
In order to implement the above process efficiently we must have a fast way to compute ∇x. We can use V to approximate this gradient by first finding the voxel v that contains x and then computing the partial derivative as the difference in cost between voxels neighboring v along each spatial dimension. We can thus compute the gradient for each $p \in P$ and arrive at the C-space gradient via Equation 5.

B. Task Space Costs

Another important class of constraints lie in the Task Space of robot, which is the space of the transform of the robot's end-effector. This space includes both the translation and rotation components and it is homeomorphic to $SE(3)$.

In previous work [21][22], we have developed Task Space Regions (TSRs), which describe sets of valid configurations in $SE(3)$. We showed that such sets are particularly useful for specifying manipulation tasks such as reaching to grasp an object or manipulating objects with pose constraints. For more complex pose constraints, we have also developed the TSR Chain representation [23], which allows TSRs to be defined relative to each other.

TSRs and TSR Chains previously defined binary constraints on end-effector pose: either the end-effector was inside the volume allowed by the TSR or it was not. We can extend these representations for cost-space planning by using the task space distance between the end-effector pose at q and the nearest TSR or TSR Chain as $C(q)$.


(a)

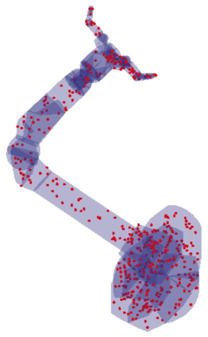

(b)

Fig. 3. (a) Example of a scene with uncertain occupancy generated using a laser-scanner. Pink: known obstacles, blue: uncertain voxels. (b) Five hundred points sampled in the volume of the Barrett arm with a bias toward the links of the hand.

Once we have this distance, the gradient can be computed using the Jacobian pseudo-inverse method. Details about computing the distance to a TSR or TSR Chain can be found in [22] and [23], respectively.

C. C-Space Costs

Finally, we consider constraints defined in the C-space. Many useful constraints can be defined in the C-space; including constraints on torque and joint limits. As an example, we present a cost function that is designed to bias the planner toward a set of known configurations. This cost function is useful for biasing the search toward a desired configuration, such as elbow-up or elbow-down, or toward a desired path.

Suppose we have a finite set of configurations U with associated costs $C(u)$ for all $u \in U$. We would like to evaluate the cost of a configuration $q \notin U$. Since we would like to bias the planner toward U, the cost should increase as we move away from U. To achieve this behavior, we propose a cost function that has the following desirable properties:

1) The function is smooth.
2) All $u \in U$ contribute to $C(q)$.
3) $C(q)$ is affected more by closer $u \in U$.

The parameters needed to define the function are the following:

- The finite set of configurations U along with $C(u)$ for all $u \in U$
- A covariance Σ for each $u \in U$

where Σ is an $n \times n$ symmetric matrix used to weight the distance metric. Thus the weighted squared distance between q and the ith configuration $u_i \in U$ is $d_i = (q-u_i)^T \Sigma_i^{-1}(q-u_i)$. The cost functions is:

$$C(q) = s \sum_{i=1}^{|U|} \left(\frac{C(u_i)}{d_i} + 1 \right) \qquad s = \left(\sum_{j=1}^{|U|} \frac{1}{d_j} \right)^{-1} \tag{6}$$

The derivative of the weighted squared distance to u_i is $d'_i = 2(q - u_i)^T \Sigma_i^{-1}$. The gradient of the cost is then

$$\nabla q = s \sum_{i=1}^{|U|} \left[\left(\frac{C(u_i)}{d_i} + 1 \right) \left(\sum_{j=1}^{|U|} \frac{d'_j}{d_j^2} \right) s - \frac{C(u_i) d'_i}{d_i^2} \right] \quad (7)$$

Note that a small offset is added to d_i when the distance approaches zero to prevent numerical instability.

VII. Example Problems

We now present three manipulation planning problems which consider different types of cost. Each problem is designed to highlight the need for an efficient method of navigating cost-space chasms. We compare the performance of the T-RRT and GradienT-RRT in the three problems for varying values of $nFailMax$ (see Figure 6) and discuss the results. We set the parameters $\Delta q_{step} = 0.05$, $initTemp = 0.01$, and $\alpha = 2$ for all experiments. The magnitude of the gradient step was bounded to be less than 0.05rad for the first and third examples and 0.1rad for the second.

A. Path Imitation using a C-Space Constraint

For many problems involving human-robot interaction, it is desirable for the paths generated by the robot to conform to some legibility or aesthetic criteria [24] that can be demonstrated via example paths. To incorporate these considerations into our planner we build on a learning-from-demonstration strategy. We first train a learning algorithm on a set of examples paths and then construct a C-space cost function around the optimal path produced by the learning algorithm. Configurations closer to the learned path receive a lower cost than configurations farther from it. This construction biases the algorithm to search around the learned path first while also allowing it to explore regions of the C-space farther from the learned path as time goes on. The advantage of this approach is that we can bias the search toward the learned path while taking into account feasibility constraints which are not accounted for by the learning algorithm, such as collision,.

In this example problem, the robot's task is to reach for a bottle on a kitchen counter (Figure 4a). Five example paths have been provided to the robot for reaching for the same bottle in different locations on the counter (Figure 5a). All of the example paths have the robot's elbow point up throughout the path (as opposed to pointing down or to the side). We use the GMM-GMR toolbox [25] to compute a set of Gaussians that describe the paths and to obtain the learned path from the Gaussians. The learned path also has associated variances in each dimension of C-space, expressing the confidence of each predicted point in each dimension (Figure 5b).

We use the cost function described in Section VI-C to transform the learned path into a cost function. The set U is the points in the path (sampled at a fixed resolution), and the Σ values are the variances of the prediction. The costs of all points in U are set to 0. The resulting cost function is shaped like a chasm in C-space with the learned path at the bottom (see Figure 5c).

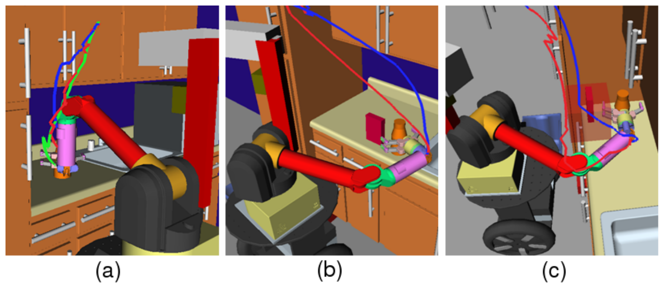


Fig. 4. Examples of paths planned for the three problems for $nFailMax$ = 30 (only the translation of the end-effector is shown) after 300 iterations of shortcut smoothing. (a) C-space path imitation, (b) Task Space constraint, (c) Uncertain workspace occupancy. Blue: T-RRT path. Red: GradienT-RRT path. The green path in (a) is the path learned from demonstration.

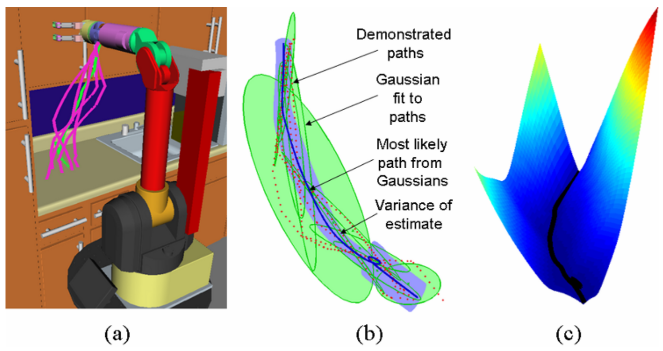


Fig. 5. (a) Example C-space paths (purple) and the learned path (green), only the translation of the end-effector is shown. (b) Process of learning the path using the GMM-GMR toolbox; the path is shown for joints five and six of the robot. (c) The learned path transformed into a cost function using Equation 6.

B. Task Space Constraints

Many common tasks in manipulation planning also involve constraints on end-effector pose. While we have treated these as hard constraints in previous work [21][22], there are also situations where they can be seen as soft constraints. Consider a reaching task where the pose of the end-effector is not constrained. We have observed that paths generated by RRTs for such problems involve significant (and often needless) rotation of the wrist. The task is accomplished but users find that the rotation of the wrist is unpredictable so they feel less comfortable around the robot.

To address such a constraint on end-effector pose, we employ TSRs or TSR Chains as soft constraints by using the task space distance to the TSR as $C(q)$. The search is thus biased to generate nodes on or near the constraint manifold defined by the TSR while still allowing the algorithm to explore beyond this manifold as time goes on.

In this example problem the task is again to reach for a bottle on the counter (Figure 4b), however we assign a TSR to bias the robot not to tilt its end-effector throughout the path. Since we are restricting two DOF of rotation, this constraint induces a lower-dimensional zero-cost manifold in the C-space; an infinitely thin chasm with cost increasing in all directions around it.

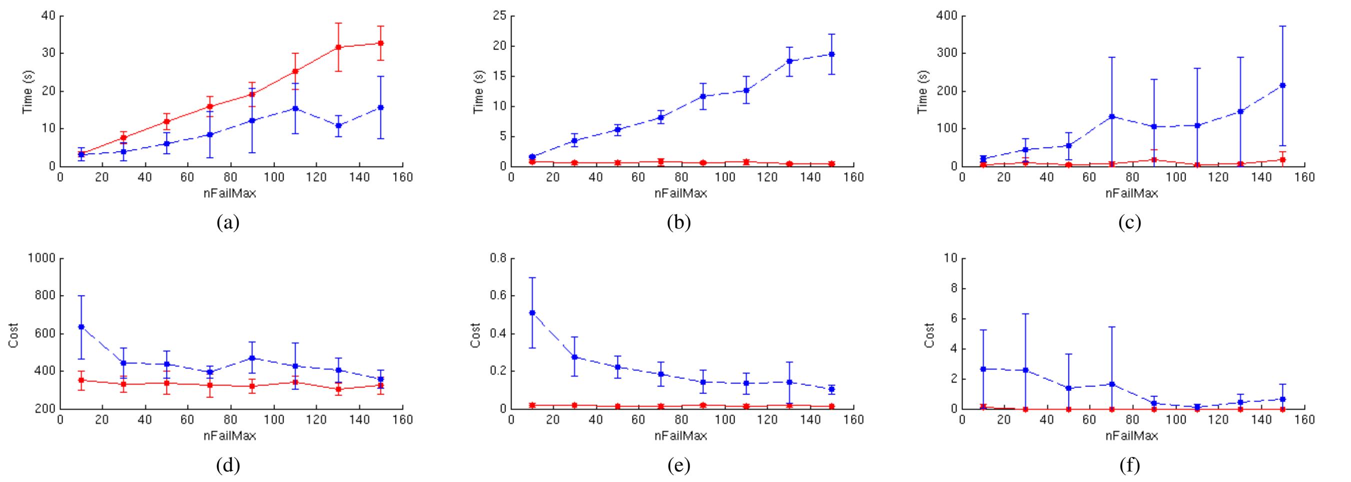


Fig. 6. (a-c) Average planning times for the C-space, task space, and workspace examples, respectively. (d-f) Corresponding average cost integral of paths after 300 iterations of shortcut smoothing. Blue Dashed: T-RRT, Red: GradienT-RRT. The error bars are standard deviations computed over 10 runs for each value of $nFailMax$.

C. *Uncertain Workspace Occupancy*

Another important use for soft constraints is in planning paths in uncertain environments, where we can not be sure if some areas of the environment are empty or filled because of incomplete sensor data. In such environments we want to encroach as little as possible on uncertain areas. While it may be tempting to treat uncertain areas as obstacles, this can lead to infeasible planning queries because, for instance, the goal configuration may intersect an uncertain area.

Our approach to this problem is to create a voxel grid over the workspace and label each voxel with its probability of occupancy. We compute the cost of a configuration of the robot from a set of points sampled within its geometry using the methods of Section VI-A. The planner is thus biased to stay away from higher-occupancy regions in the workspace during the path and to escape from these regions without increasing cost if the start or goal lies within one of them. However, given more time the algorithm will begin to explore the higher-occupancy regions as well. Note that we still impose hard collision constraints for known obstacles in addition to this constraint.

In this example problem, the task is to reach for a bottle in a cluttered space (Figure 4c). An uncertain region has been placed in front of the robot. It is possible for the robot to avoid the uncertain region completely by reaching around it, but this induces a narrow passage in workspace between the uncertain region and the boundary of the robot's reachability. Thus the constraint is a one-sided chasm, with the workspace closer to the base of the robot being higher cost and the workspace farther from the robot being unreachable.

We also implemented a similar scenario on the HERB robot, where the uncertainty in the environment is derived from laser-scan data (Figure 7). Here the bottle is in a cabinet which is occluded from the laser scanner by the cabinet door. Note that the bottle is engulfed by the uncertainty created by this occlusion so the robot cannot simply avoid the uncertain region when reaching to grasp. Please see our video for the execution of the paths planned by GradienT-RRT (several snapshots are shown in Figure 7).

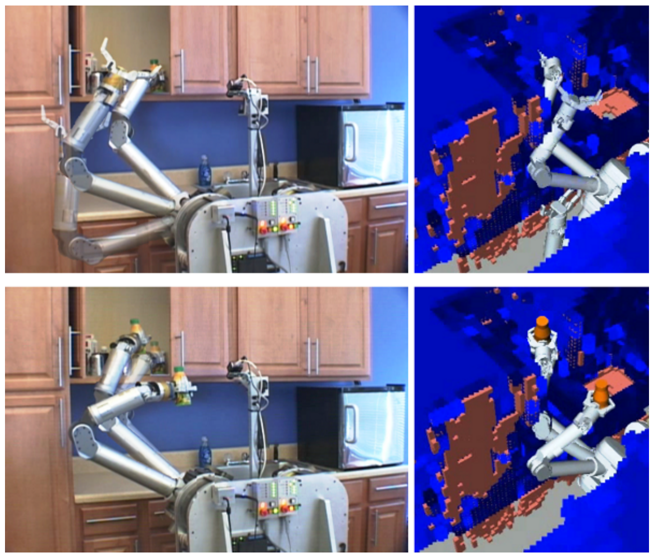


Fig. 7. Time-lapse images from the execution of two paths planned by GradienT-RRT with uncertain workspace occupancy. The blue areas in the robot's world model (right) represent the probability of occupancy of a cell, with light blue being less occupied and dark blue being more occupied.

D. *Discussion of Results*

Comparisons of run time and path integral cost are shown in Figure 6. In all three examples, the costs of the paths produced by GradienT-RRT were lower on average than those produced by the T-RRT for all values of $nFailMax$. The computation times of GradienT-RRT were much lower than those for T-RRT for the task space and uncertainty examples. The path imitation example's computation time is higher (Figure 6a) because the learned path collides with the upper cabinet and the bottle; i.e. the chasm intersects an obstacle (as in Figure 2b). As a result the GradienT-RRT attempts to generate more nodes in the chasm, which slows down the algorithm at higher values of $nFailMax$.

What is most encouraging about these results is that the cost of the path produced by GradienT-RRT only decreases slightly when increasing $nFailMax$. The GradienT-RRT is far less sensitive to the $nFailMax$ parameter because it does not rely on sampling to traverse a chasm; it uses the gradient to generate nodes in low cost regions. Thus we can set $nFailMax$ to be relatively low (between 30 and 50) and still achieve roughly the same performance as setting it higher. We believe that this is a significant improvement over T-RRT, where increasing $nFailMax$ is the only way to compute better paths but doing so produces an increase in computation time.

VIII. Conclusion and Future Work

We have presented the GradienT-RRT algorithm as a method to navigate chasms in cost space. The algorithm is a combination of the T-RRT and local gradient descent. Because gradient-descent does not rely on sampling to find lower-cost configurations, GradienT-RRT can traverse cost-space chasms more quickly and at lower cost than T-RRT. We have also described general cost functions in workspace, task space, and C-space that are useful for manipulation planning and that induce such cost-space chasms. When we compared GradienT-RRT and T-RRT on three example problems, we found that the GradienT-RRT outperformed T-RRT in terms of the cost of the final path for all examples and in terms of computation time for two of the three examples. We also found that GradienT-RRT was not very sensitive to the $nFailMax$ parameter, making it easier to tune than T-RRT. Finally we showed GradienT-RRT running on the physical HERB robot, where it planned to retrieve an object while considering uncertain workspace occupancy.

In future work we would like to explore planning with multiple simultaneous constraints. Specifically, we are interested in ways to optimize multiple constraints without mixing them into a single cost function. One of the key challenges is combining multiple gradients into a single direction vector.

We would also like to address the issue of goal sampling so that GradienT-RRT can generate goals during the planning process instead of being restricted to a single goal. This is important because there is often an infinite set of goal configurations in many types of manipulation problems.

IX. Acknowledgements

Dmitry Berenson was supported by LAAS-CNRS and by NSF Grant No. EEC-0540865. Thanks to Jim Mainprice, Romain Iehl, and Nathan Ratliff for helpful discussions.

References

[1] L. Jaillet, J. Cortés, and T. Siméon, "Sampling-Based Path Planning on Configuration-Space Costmaps," *IEEE Transactions on Robotics*, vol. 26, no. 4, pp. 635–646, Aug. 2010.

[2] J. Kuffner, J. J. and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[3] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2003.

[4] T. Sugihara and Y. Nakamura, "Whole-body cooperative balancing of humanoid robot using cog jacobian," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[5] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Transactions on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[6] S. Sentis and O. Khatib, "Synthesis of whole-body behaviors through hierarchical control of behavioral primitives," *International Journal of Humanoid Robotics*, vol. 2, pp. 505–518, December 2005.

[7] D. Ferguson and A. Stentz, "Anytime RRTs," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[8] A. Ettlin and H. Bleuler, "Randomised Rough-Terrain Robot Motion Planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[9] M. Apaydin, A. Singh, D. Brutlag, and J.-C. Latombe, "Capturing molecular energy landscapes with probabilistic conformational roadmaps," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2001.

[10] P. Chen and Y. Hwang, "SANDROS: a dynamic graph search algorithm for motion planning," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 3, pp. 390–403, Jun 1998.

[11] R. Geraerts and M. H. Overmars, "Creating High-quality Paths for Motion Planning," *International Journal of Robotics Research*, vol. 26, no. 8, pp. 845–863, 2007.

[12] S. Rodriguez and N. Amato, "An obstacle-based rapidly-exploring random tree," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[13] J. Pan, L. Zhang, and D. Manocha, "Retraction-based RRT planner for articulated models," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2010.

[14] D. Hsu and J. Reif, "The bridge test for sampling narrow passages with probabilistic roadmap planners," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[15] S. Dalibard and J. Laumond, "Control of probabilistic diffusion in motion planning," in *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2009.

[16] D. Hsu, G. Sanchez-Ante, H.-l. Cheng, and J.-C. Latombe, "Multi-level free-space dilation for sampling narrow passages in PRM planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[17] L. Kavraki, J. Latombe, R. Motwani, and P. Raghavan, "Randomized Query Processing in Robot Path Planning," *Journal of Computer and System Sciences*, vol. 57, no. 1, pp. 50–60, Aug. 1998.

[18] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[19] J. Ratcliff. (2010) Convex decomposition library. [Online]. Available: http://code.google.com/p/convexdecomposition/

[20] L. Devroye, *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, 1986, pp. 567–571.

[21] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. Kuffner, "Manipulation planning on constraint manifolds," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[22] D. Berenson, S. S. Srinivasa, D. Ferguson, A. Collet, and J. Kuffner, "Manipulation planning with workspace goal regions," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[23] D. Berenson, J. Chestnutt, S. S. Srinivasa, J. J. Kuffner, and S. Kagami, "Pose-Constrained Whole-Body Planning using Task Space Region Chains," in *Humanoids*, 2009.

[24] E. A. Sisbot, L. F. Marin-Urias, R. Alami, and T. Simeon, "Human aware mobile robot motion planner," *IEEE Transactions on Robotics*, vol. 23, pp. 874–883, 2007.

[25] S. Calinon, F. Guenter, and A. Billard, "On Learning, Representing, and Generalizing a Task in a Humanoid Robot," *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 2, pp. 286–298, Apr. 2007.