Addressing Pose Uncertainty in Manipulation Planning Using Task Space Regions

Dmitry Berenson[†] Siddhartha S. Srinivasa[‡†] James J. Kuffner[†]

[†]*The Robotics Institute, Carnegie Mellon University*
5000 Forbes Ave., Pittsburgh, PA, 15213, USA
[dberenso, kuffner]@cs.cmu.edu

[‡]*Intel Research Pittsburgh*
Pittsburgh, PA, 15213, USA
[siddhartha.srinivasa]@intel.com

***Abstract*—We present an efficient approach to generating paths for a robotic manipulator that are collision-free and guaranteed to meet task specifications despite pose uncertainty. We first describe how to use Task Space Regions (TSRs) to specify grasping and object placement tasks for a manipulator. We then show how to modify a set of TSRs for a certain task to take into account pose uncertainty. A key advantage of this approach is that if the pose uncertainty is too great to accomplish a certain task, we can quickly reject that task without invoking a planner. If the task is not rejected we run the IKBiRRT planner, which trades-off exploring the robot's C-space with sampling from TSRs to compute a path. Finally, we show several examples of a 7-DOF WAM arm planning paths in a cluttered kitchen environment where the poses of all objects are uncertain.**

I. Introduction

A common assumption when planning for robotic manipulation tasks is that the robot has perfect knowledge of the geometry and pose of objects in the environment. For a robot operating in a home environment it may be reasonable to have geometric models of the objects the robot manipulates frequently and/or the robot's work area. However, these objects and the robot frequently move around the environment, introducing uncertainty into the pose of the objects relative to the robot. Laser-scanners, cameras, and sonar sensors can all be used to help resolve the poses of objects in the environment, but these sensors are never perfect and usually localize the objects to be within some hypothetical probability distribution of pose estimates. If this set of probable poses is large, planning with the best hypothesis alone can be unsafe because the robot could collide with a poorly-localized object (see Figure 1).

In addition to being unsafe in terms of potential collisions, planning without regard to uncertainty can violate task specifications. Suppose that a robot arm is to pick up an object by placing its end-effector at a particular pose relative to the object and closing the fingers. If there is any uncertainty in the pose of the object, generally no guarantee can be made that the end-effector will reach a specific point relative to the true pose of object. Depending upon the task, this lack of precision may range from being the source of critical failure to causing only minor disturbances. Many practical manipulation tasks afford a large amount of freedom in the choice of grasps and goal locations. For example, when we pick up a coffee mug and place it in the sink, we can choose from a wide range of hand configurations to grasp the mug securely, as well as a wide range of goal locations in the sink to place the mug.

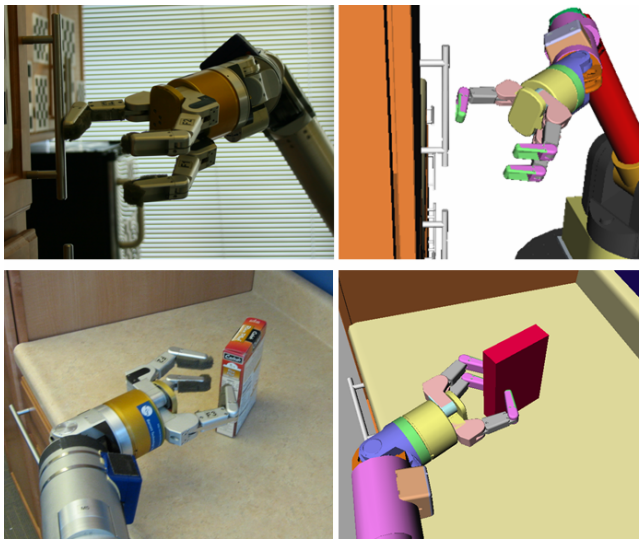

Fig. 1. Failures arising from planning with the best single hypothesis under uncertainty. *Top:* An unexpected collision with the environment due to uncertain robot localization. *Bottom:* A grasp fails due to uncertain object pose estimation. In both cases, planning with only the best estimate of pose incorrectly predicts success (*right*).

In this paper we employ the concept of Task Space Regions (TSRs) [1] to meet task specifications in spite of pose uncertainty. TSRs allow the specification of continuous regions in $SE(3)$, the six-dimensional space of poses, as goals for a manipulator's end-effector. A given task can entail any number of TSRs, each of which encompasses a subspace of $SE(3)$, with dimension less than or equal to six. TSRs allow us to plan for manipulation tasks in the presence of pose uncertainty by ensuring that the given task requirements are satisfied for all hypotheses of the object's pose. TSRs also provide a way to quickly reject tasks which cannot be guaranteed to be accomplished given the current pose uncertainty estimates.

In the following sections we motivate and describe TSRs and show how they can be incorporated into sampling-based planning algorithms. We then show how to modify the TSRs of a given task to account for pose uncertainty. We also describe the Inverse Kinematics BiDirectional RRT algorithm (IKBiRRT), which uses TSRs to plan paths for manipulation tasks. Finally, we demonstrate our methods on the 7DOF WAM arm performing manipulation tasks in a kitchen environment.

II. Background

The idea of motion planning in the presence of uncertainty dates back to the seminal work of Lozano-Perez *et. al.* [2] on *preimage backchaining*. The concept of a preimage — a region of configuration space from which a motion command is guaranteed to attain a given goal recognizably — was used as a building block to compose an operational planner that produced actions guaranteed to succeed under pose and action uncertainty. However, it was shown [3], [4] that constructing preimages incurred a prohibitive computational cost. We will show in Section IV that, for the case of manipulation planning, TSRs provide an efficient representation for computing the preimage of hand poses that are guaranteed to provide an acceptable grasp or object placement under object pose uncertainty.

Our planning algorithm, the IKBiRRT [1], is based on the bidirectional variant of the RRT [5], [6]. This algorithm uses a backward-searching tree that has multiple roots. However, unlike previous work [7], [8], these roots are added to the tree during the planning process and are guaranteed to meet task specifications despite pose uncertainty.

Another area of related work is quality metrics that characterize the robustness of grasps to pose uncertainty. Techniques for computing such metrics have ranged from first and second-order analysis of the perturbation of grasp contact points [9], [10], [11], [12] to generating contact patches [12], [13], [14]. These metrics can be used to construct robust TSRs for grasping under uncertainty.

Our approach requires that we have a set of hypotheses of pose for every object in the environment other than the robot. These hypothesis sets can come from sensors that are on-board the robot, sensors that are fixed in the environment, or a mixture of both. In general, pose estimates can be propagated through different frames using the methods of Smith and Cheeseman[15]. Though Smith and Cheeseman focus mainly on 2D poses, their methods have been extended to six-dimensional poses in 3D. See Sallinen's thesis[16] for an overview.

III. Task Space Regions

In general, a set of task space goals for a manipulator's end-effector can be defined as comprising any number of continuous regions of $SE(3)$ of arbitrary size and shape distributed throughout the task space. However, such a broad representation lacks three fundamental properties that are important for efficient sampling-based planning under uncertainty.

First, the set of goals must be relatively easy to specify for the user. Consider the task of placing an object onto a table. The set of all valid end-effector positions that achieve the placement of the object produces a complex volume which may be infeasible and/or computationally expensive to discretize and input into a planner.

Second, sampling from the set of goals must be efficient and ideally cover the entire goal set. If the set is discretized, some interpolation scheme between points must be used for sampling, which could involve greater computational cost.

Third, in the presence of pose uncertainty, it is practically impossible to guarantee that a certain point relative to an object will be reached by the end-effector. Thus there is no guarantee that moving the end-effector to a point in the goal set will result in meeting the task specification.

In previous work we introduced TSRs [1], which approach the problem of specifying goal sets by describing implicit volumes of $SE(3)$. These volumes are particularly useful for manipulation tasks such as reaching to grasp an object or placing an object onto some 2D surface or into some 3D volume. TSRs are also intuitive to specify, can be efficiently sampled, and the TSRs of a given task can be modified to take into account pose uncertainty. A set of TSRs can describe any arbitrary set of goals by, in the extreme case, assigning one TSR to every point, though this is clearly undesirable in practice because the specification, sampling, and uncertainty problems will re-emerge. However, for the types of grasping and object placement problems we are interested in, we typically need to use less than 20 TSRs to specify our task space goals adequately.

TSRs specify goal poses of a robot's end-effector but they can equivalently specify the goal placements of an object that the robot is holding. Thus we will consider these two tasks equivalent in the rest of this paper.

A. TSR Definition

Throughout this paper, we will be using transformation matrices of the form $\mathbf{T}_b^a \in SE(3)$, which specifies the pose of b in the coordinates of frame a. $\mathbf{T}_b^a$, written in homogeneous coordinates, consists of a 3×3 rotation matrix $\mathbf{R}_b^a$ and a 3×1 translation vector $\mathbf{t}_b^a$.

$$\mathbf{T}_b^a = \begin{bmatrix} \mathbf{R}_b^a & \mathbf{t}_b^a \\ \mathbf{0} & 1 \end{bmatrix} \tag{1}$$

A TSR consists of three parts:

- $\mathbf{T}_w^0$: reference transform of the TSR in world coordinates
- $\mathbf{T}_e^w$: end-effector offset transform in the coordinates of w
- $\mathbf{B}^w$: 6×2 matrix of bounds in the coordinates of w:

$$\mathbf{B}^w = \begin{bmatrix} x_{min} & x_{max} \\ y_{min} & y_{max} \\ z_{min} & z_{max} \\ \psi_{min} & \psi_{max} \\ \theta_{min} & \theta_{max} \\ \phi_{min} & \phi_{max} \end{bmatrix} \tag{2}$$

The first three rows of $\mathbf{B}^w$ bound the allowable translation along the x, y, and z axes (in meters) and the last three bound the allowable rotations about those axes (in radians), all in the w frame. Note that this assumes the Roll-Pitch-Yaw (RPY) Euler angle convention, which is used because it allows bounds on rotation to be specified intuitively.

In practice, the w frame is usually centered at the origin of an object held by the hand or at a location on an object that is useful for grasping. We use an end-effector offset transform $\mathbf{T}_e^w$, because we do not assume that w directly encodes the

pose of the end-effector. $\mathbf{T}_e^w$ allows the user to specify an offset from w to the origin of the end-effector e, which is extremely useful when we wish to specify a TSR for an object held by the hand or a grasping location which is offset from e; for instance in between the fingers. For some example $\mathbf{T}_e^w$ transforms, see Figure 5.

B. Sampling From Task Space Regions

In order to use TSRs in a bidirectional sampling-based planner like the IKBiRRT, we must be able to sample from a set of TSRs efficiently. The IKBiRRT uses these samples as queries for an inverse-kinematics solver.

Sampling from a single TSR is done by first sampling a random value between each of the bounds defined by $\mathbf{B}^w$ with uniform probability. These values are then compiled in a displacement $d^w_{sample} = [x\ y\ z\ \psi\ \theta\ \phi]$ and converted into the transformation $\mathbf{T}^w_{sample}$. We can then convert this sample into world coordinates after applying the end-effector transformation.

$$\mathbf{T}^0_{sample'} = \mathbf{T}^0_w \mathbf{T}^w_{sample} \mathbf{T}^w_e \tag{3}$$

We observe that while our method ensures a uniform sampling in the bounds of $\mathbf{B}^w$, it could likely produce a biased sampling in the subspace of constrained spatial displacements $SE(3)$ that $\mathbf{B}^w$ parameterizes. We are currently investigating how to generate an unbiased sampling, however this bias has not had a significant impact on the runtime or success-rate of our algorithm.

In the case of multiple TSRs define for a single task, we must first decide which TSR to sample from. If all TSRs enclose six-dimensional volumes, we can choose among TSRs in proportion to their volume. However a volume-proportional sampling will ignore TSRs that encompass volumes of less than six dimensions because they have no volume in the six-dimensional space. To address this issue we propose a weighted sampling scheme that samples TSRs proportional to the *sum* of the differences between their bounds.

$$\zeta_i = \sum_{j=1}^{6} \left(\mathbf{B}^{w_i}_{j,2} - \mathbf{B}^{w_i}_{j,1} \right) \tag{4}$$

where ζ_i and $\mathbf{B}^{w_i}$ are the weight and bounds of the ith TSR, respectively. Sampling proportional to ζ_i allows us to sample from TSRs of any dimension except 0 while giving preference to TSRs that encompass more volume. TSRs of dimension 0, i.e. points, are given an ϵ probability of being sampled. In general, any sampling scheme for selecting a TSR can be used as long as there is a non-zero probability of selecting any TSR.

IV. Accounting For Pose Uncertainty

Since we would like to compute a plan that is guaranteed to meet task specifications for all hypotheses of object pose, we must modify the TSRs assigned to a given task to account for pose uncertainty and introduce new obstacles into the world to avoid potential collisions.

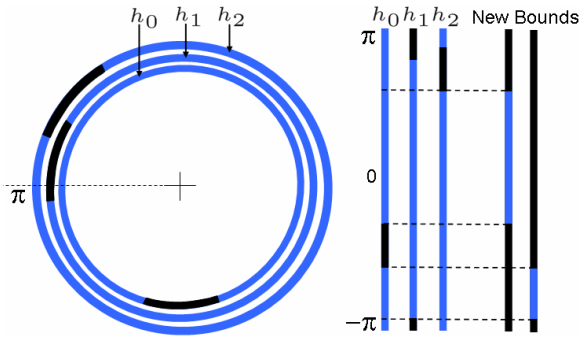

Fig. 2. Process for splitting TSRs to take into account rotation. Only one dimension of rotation is shown here. The three concentric circles correspond to a *single* TSR's bound in Roll that has been rotated by transforms $\mathbf{T}^0_{h_0}$, $\mathbf{T}^0_{h_1}$, and $\mathbf{T}^0_{h_2}$. Blue regions correspond to allowable rotations and black ones to unallowable rotations. The circles are cut at $\pi = -\pi$ and overlaid on the right. The strips where all rotations are valid (there are no black regions) are extracted as new separate bounds for this dimension. This process is identical for Roll, Pitch, and Yaw. The cartesian product of the new bounds for Roll, Pitch, and Yaw along with the original x, y, and z bounds produces a new set of TSRs $\mathcal{T}_{split}$.

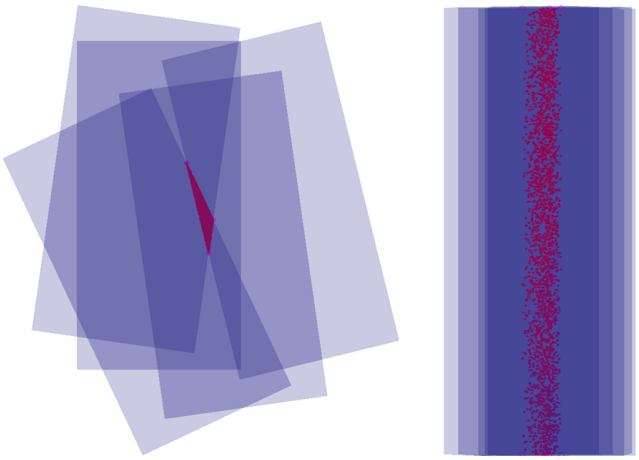

Fig. 3. Intersection of five instances of a TSR. *Left*: x-y view. *Right*: y-z view. The red points are sampled within the polytope of intersection using rejection sampling.

A. Uncertainty and TSRs

Let the set of pose hypotheses for a given object be a set of transformation matrices $\mathcal{H}$. Also, let the set of TSRs defined for this object be $\mathcal{T}$. The process for generating a new set of TSRs $\mathcal{T}_{new}$ that takes into account $\mathcal{H}$ is shown in Algorithm 1.

This algorithm first splits every TSR $t \in \mathcal{T}$ to take into account the rotation uncertainty in $\mathcal{H}$, generating a set $\mathcal{T}_{split}$ for each t. See Figure 2 for an illustration of this process. It then places a duplicate of each $t_s \in \mathcal{T}_{split}$ at every location defined by the transforms in $\mathcal{H}$. Next, it computes the volume of intersection of all duplicates for every t_s (see Figure 3). This is done by converting all faces of all duplicates into linear constraints via the FacesToLinInequalities function and then converting those linear constraints into vertices P of a 6D polytope via the GetVerticesInequalities function. Since TSRs are convex we know that the polytope of intersection must be convex as well. If the uncertainty is too great (i.e. there is no 6D point where all duplicates intersect), P will be empty. If P is not empty, we place an axis-aligned bounding box around

P, set this as the new bounds of t_s, and add t_s to $\mathcal{T}_{new}$. Note that it is irrelevant which element of $\mathcal{H}$ is used as $\mathbf{T}^0_{h_0}$ because the results will always be the same in the world frame.

In order to guarantee that a sampled 6D point meets the uncertainty specification of the problem, samples drawn from t_s must lie inside the polytope defined by P. Ideally, we would like to generate uniformly random samples from within P directly. Indeed, this is always possible because the polytope defined by P is convex. Because the polytope is convex, it can always be divided into simplices using Delaunay Triangulation. To generate a uniformly random sample from a collection of simplices, we first select a simplex proportional to its area and then sample within that simplex by generating a random linear combination of its vertices [17]. For simple polytopes, this method is quite efficient, however as the polytope defined by P grows more complex, the Delaunay Triangulation becomes more costly, thus this method usually does not scale well with the number of hypotheses in $\mathcal{H}$.

Rejection sampling can also be used to sample from the polytope defined by P. When using rejection sampling, we sample a point x uniformly at random from the bounding-box of P until we find an x which satisfies $b - Ax \geq 0$. This method is quite fast in practice and does not require triangulating the polytope defined by P, thus it is more suitable for use in an online planning scenario.

In order to accommodate rejection sampling with TSRs, we add another element to our TSR definition:

- **LI**: Linear inequalities of the form $b - Ax \geq 0$

If $\mathbf{LI} = \emptyset$ (i.e. there is no uncertainty), any sample generated within the bounding box is valid. If the $\mathcal{T}_{new}$ returned by ApplyUncertainty($\mathcal{T}$, $\mathcal{H}$) is empty, then we know that it is impossible to accomplish this task with the uncertainty in $\mathcal{H}$. We can thus reject this task without calling the planner, a key advantage of our approach.

B. Avoiding Potential Collisions

The process for avoiding potential collisions is quite straightforward. For every object, we have a set of hypothesis poses $\mathcal{H}$. To account for potential collisions with a given object, we simply create a duplicate of that object for each pose in $\mathcal{H}$ and place it at that pose in our simulation environment. This is done independently of the TSR modification described above.

Note that these duplications will slow down the planning process because multiple collision checks must be performed for a single object. If the hypotheses in $\mathcal{H}$ are drawn from a convex set (say an ellipse), we may instead consider computing the swept-volume of the object as it moves between different hypotheses. However, swept-volume computations are known to be quite expensive and we do not wish to restrict our algorithm to only hypotheses from convex sets. This is because many pose estimation algorithms (e.g. particle filters) can give non-convex or multi-modal estimates of pose. Thus to preserve generality, we follow the duplication procedure described above despite the added cost of collision-checking with multiple instances of the same object.

Algorithm 1: ApplyUncertainty($\mathcal{T}$,$\mathcal{H}$)

$\mathbf{T}^0_{h_0} \leftarrow$ Any element of $\mathcal{H}$;
$\mathcal{T}_{new} \leftarrow \emptyset$;
for $t \in \mathcal{T}$ **do**
  $\mathcal{T}_{split} \leftarrow$ SplitRotations(t, $\mathbf{T}^0_{h_0}$, $\mathcal{H}$);
  for $t_s \in \mathcal{T}_{split}$ **do**
    $A \leftarrow \emptyset$; $b \leftarrow \emptyset$;
    for $\boldsymbol{T}^0_h \in \mathcal{H}$ **do**
      $V \leftarrow$ GetVertices(t_s);
      $V_{xyz} \leftarrow (\mathbf{T}^0_{h_0})^{-1}\mathbf{T}^0_h V_{xyz}$;
      $F \leftarrow$ GetFaces(V);
      $\{A_{temp}, b_{temp}\} \leftarrow$ FacesToLinInequalities(F);
      $A \leftarrow A \cup A_{temp}$;
      $b \leftarrow b \cup b_{temp}$;
    end
    $P \leftarrow$ GetVerticesFromInequalities(A, b);
    if $P = \emptyset$ **then**
      $t_s.\mathbf{T}^0_w \leftarrow h_0$;
      $t_s.\mathbf{B}^w \leftarrow$ BoundingBox(P);
      $t_s.\mathbf{T}^w_e \leftarrow t.\mathbf{T}^w_e$;
      $t_s.\mathbf{LI} \leftarrow \{A, b\}$;
      $\mathcal{T}_{new} \leftarrow \mathcal{T}_{new} \cup t_s$;
    end
  end
end
return $\mathcal{T}_{new}$;

V. The IKBiRRT Algorithm

Once we have properly taken into account uncertainty by generating a new set of TSRs and augmenting our simulation environment with duplicates of obstacles, we run the IKBiRRT planner (first presented in [1]) to find a C-space path for the robot that brings its end-effector (or the object it is holding) to a pose within one of our new TSRs.

The IKBiRRT (see Algorithm 2) is an extension of the Bidirectional RRT (BiRRT) algorithm [18] that grows trees from both the start and goal configurations. At each iteration, IKBiRRT chooses between one of two modes: exploration of the C-space using a standard BiRRT and sampling from the set of TSRs $\mathcal{T}$. The probability of choosing the sampling mode is controlled by the parameter P_{sample}.

The Extend function moves incrementally from a given starting configuration toward a target configuration in fixed step sizes, stopping only when it encounters a collision or when the target configuration has been reached. The nodes generated through this process are added to the tree passed in to the Extend function. If both trees meet at some configuration, a path from the start to a goal configuration has been found and the algorithm extracts the path and smooths it. Smoothing is performed using the shortcut heuristic as in [19]; however, any smoothing method is acceptable.

The AddIKSolutions function injects goal configurations into the backward tree T_{goal}. To do this, we first sample a

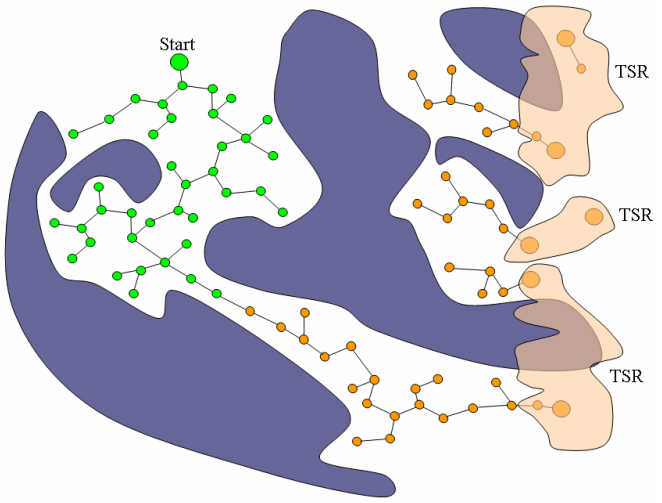

Fig. 4. Depiction of the IKBiRRT planning in the robot's C-space with multiple TSRs. The blue regions are C-space obstacles, the forward-searching tree is shown with green nodes, and the backward-searching tree is shown with orange nodes. The large orange nodes (roots of the backward-searching tree) are derived from TSR samples that are generated as the planner is running.

Algorithm 2: IKBiRRT(q_s, $\mathcal{T}$)

```
1  T_a.Init(q_s); T_b.Init(NULL);
2  while TimeRemaining() do
3      T_goal = GetBackwardTree(T_a, T_b);
4      if T_goal.size = 0 or rand(0,1) < P_sample then
5          AddIKSolutions(T_goal, T);
6      else
7          q_rand ← RandConfig();
8          q^a_near ← NearestNeighbor(T_a, q_rand);
9          q^a_reached ← Extend(T_a, q^a_near, q_rand);
10         q^b_near ← NearestNeighbor(T_b, q^a_reached);
11         q^b_reached ← Extend(T_b, q^b_near, q^a_reached);
12         if q^a_reached = q^b_reached then
13             P ← ExtractPath(T_a, q^a_reached, T_b, q^b_reached);
14             return SmoothPath(P);
15         else
16             Swap(T_a, T_b);
17         end
18     end
19 end
20 return NULL;
```

point in $\mathcal{T}$ using the method described in Section III-B along with the rejection sampling described in Section IV-A, which gives us the transform $\mathbf{T}^0_{sample'}$. This transform is passed to the IK solver of the given manipulator. The manipulator then generates some number of IK solutions for this transform and checks each one for collision. The collision-free solutions are added as goal configurations into T_{goal}. Note that no extra processing is needed to manage the multiple goal configurations, they are simply added as root nodes in the backward tree and treated the same as all other nodes in the tree when computing nearest-neighbors.

Probabilistic completeness of the IKBiRRT algorithm follows from the property that, as time goes to infinity, every measurable ball in the manifolds of configurations corresponding to the TSRs will be sampled and a corresponding node will be added to the backward search tree. This is because, as time goes to infinity, every measurable ball in the TSRs will be sampled, and the sample's IK solutions will be added to the backward tree. As long as the IK solver used is also probabilistically complete (i.e. it does not exclude any measurable ball of IK solution as the number of times it is invoked approaches infinity), this guarantees that no measurable ball of goal configurations will be excluded from the backward search tree. Since the BiRRT algorithm is also probabilistically complete, as time goes to infinity the forward and backward trees will connect and return a solution involving one of the goal configurations if a solution exists.

A. Generating IK Solutions

The IKBiRRT relies on the ability of the IK solver to quickly generate solutions when given a target transform for the manipulator's end-effector. In theory, a general IK solver based on the Jacobian pseudo-inverse or Jacobian-transpose methods [20] can accomplish this task, however we have found that such solvers frequently encounter problems with joint limits and that they often require many iterations, and thus significant computation time, to converge. For 6DOF manipulators such as the Puma arm, an analytical solution to the inverse kinematics problem is available and ideal for the IKBiRRT. However, for redundant robots such the 7DOF WAM arm, there are a potentially infinite number of IK solutions for a given end-effector transformation and no analytical algorithm can be used. To deal with this issue we use a *pseudo-analytical* IK solver, which discretizes the first joint of the WAM arm into a series of joint positions and computes the analytical IK solutions for the remaining 6DOF for each of these joint positions.

Note that such a strategy preserves the probabilistic completeness of the IKBiRRT because, as time goes to infinity, every IK solution (up to the discretization used in the pseudo-analytical solver) will be generated for every point in $\mathcal{T}$.

VI. Results

To evaluate our approach, we conducted experiments to measure the time taken to apply uncertainty to TSRs (Algorithm 1) and to plan paths for a 7 DOF WAM arm in a cluttered kitchen environment. The TSRs in these experiments were defined for a juice bottle (orange bottle) and rice box (red box) shown in Figure 5.

The juice bottle has a TSR that allows the hand to spin about the z-axis of the bottle and also several centimeters of freedom in translation. The $\mathbf{T}^w_e$ is set such that the fingers envelope the bottle. The bounds of this TSR are:

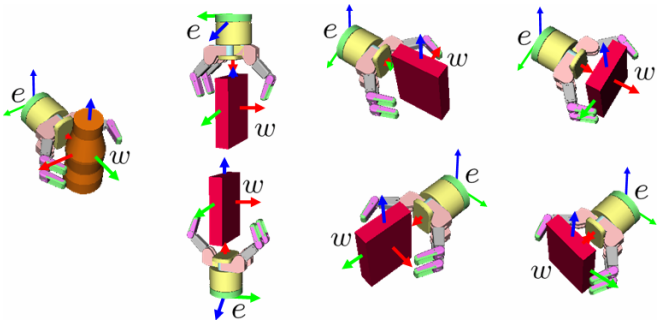

Fig. 5. Depiction of the w and e frames which are used to get the $\mathbf{T}_w^0$ and $\mathbf{T}_e^w$ transforms for the juice bottle and rice box when reaching to grasp. We also include copies of these TSRs with the hand rotation by πrad around the x axis (*red*).

$$\mathbf{B}^w = \begin{bmatrix} -0.02 & 0.02 \\ -0.02 & 0.02 \\ -0.02 & 0.02 \\ 0 & 0 \\ 0 & 0 \\ -\pi & \pi \end{bmatrix} \quad (5)$$

We also add another TSR with identical bounds, but with $\mathbf{T}_e^w$ defined to flip the hand.

For the rice box, we start with six TSRs, one for each face. The $\mathbf{B}^w$ is set according to the dimension of each face and $\mathbf{T}_e^w$ for each TSR points the hand toward the corresponding face. We also allow ± 0.4rad rotation freedom about the z-axis of the rice box in each $\mathbf{B}^w$. As with the juice bottle, we add 6 more TSRs for that are identical to the previous 6 except that $\mathbf{T}_e^w$ flips the hand.

A. Applying Uncertainty to TSRs

We conducted several experiments to gauge the performance of our algorithm for applying uncertainty to TSRs with various numbers of pose hypotheses. The pose hypotheses were sampled uniformly from ± 1.5cm in x and y and ± 0.2rad in Yaw. There was no error in the other dimensions because all the objects we detected to lie on a flat surface. In practice, we would project pose hypotheses generated by a sensor onto the flat surface to eliminate error in the other dimensions.

The goal was to see how the necessary runtime scaled with respect to the number of pose hypotheses. The results are shown in Table I. We found that the runtime scaled approximately linearly with the number of hypotheses. The runtime for the rice box was greater because it has 12 associated TSRs whereas the juice bottle only has 2. We are currently re-implementing this process to make it more efficient.

No. Hypotheses	1	15	30	45	60
Juice Bottle	0	0.17	0.29	0.44	0.58
Rice Box	0	0.85	1.60	2.40	3.10

TABLE I: RUNTIMES(IN SECONDS) FOR APPLYING UNCERTAINTY

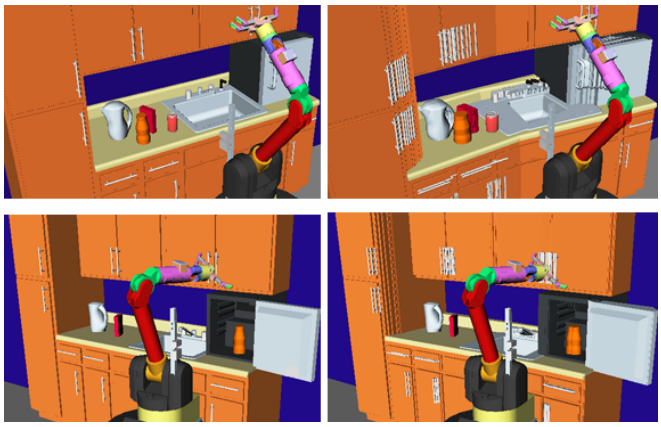

Fig. 6. Scenes 1 (*top*) and 2 (*bottom*) before duplication (*left*) and after duplication (*right*). The orange bottle is the juice bottle and the red box is the rice box.

B. Reaching in Cluttered Environments

Our approach can be used to plan motion for a reach-to-grasp of a certain object or for placing an object already held by the robot onto some surface or into some volume. However, in practice, reaching tasks are usually more constrained because the TSRs defined for grasping an object are usually far smaller than those for placing it. Thus we demonstrate our approach on reaching tasks to show its ability to work with more stringent constraints.

Three experiments were conducted: reaching for the juice bottle in scene 1, reaching for the rice box in scene 1, and reaching for the juice in scene 2 (see Figure 6). The first task is fairly easy because the juice bottle is in a relatively open area. the second task is harder because the robot's arm must squeeze between two objects to grasp the box. If there were no uncertainty, the robot would simply reach for the box from the top, however the uncertainty in the pose of the box invalidates the TSR that allows approaching from the top. The remaining TSRs only allow the robot to approach toward the thin edge of the box. The third task is the most difficult because the duplication of the refrigerator leaves very little room for the robot to approach the juice bottle. See Figure 7 for typical results of the three experiments. The results of 10 runs of each experiment for varying numbers of pose hypotheses for each object are shown in Table II.The run time was limited to 1 minute of a Dual-Core 3GHz machine with 4GB or RAM. If any run exceeded the limit, it was marked as a failure and 1 minute was used in computing the average runtime. $P_{sample} = 0.25$ for all experiments.

The object pose hypotheses were sampled uniformly from ± 2cm in x and y and ± 0.1rad in Yaw for Scene 1. However, this proved to be too much uncertainty for the task in scene 2 (the duplicates of the refrigerator made the task infeasible). Thus, in scene 2, we used an error of ± 1cm in x and y and ± 0.05rad in Yaw. This is an instance of different tasks requiring different degrees of certainty to guarantee that the task is accomplished.

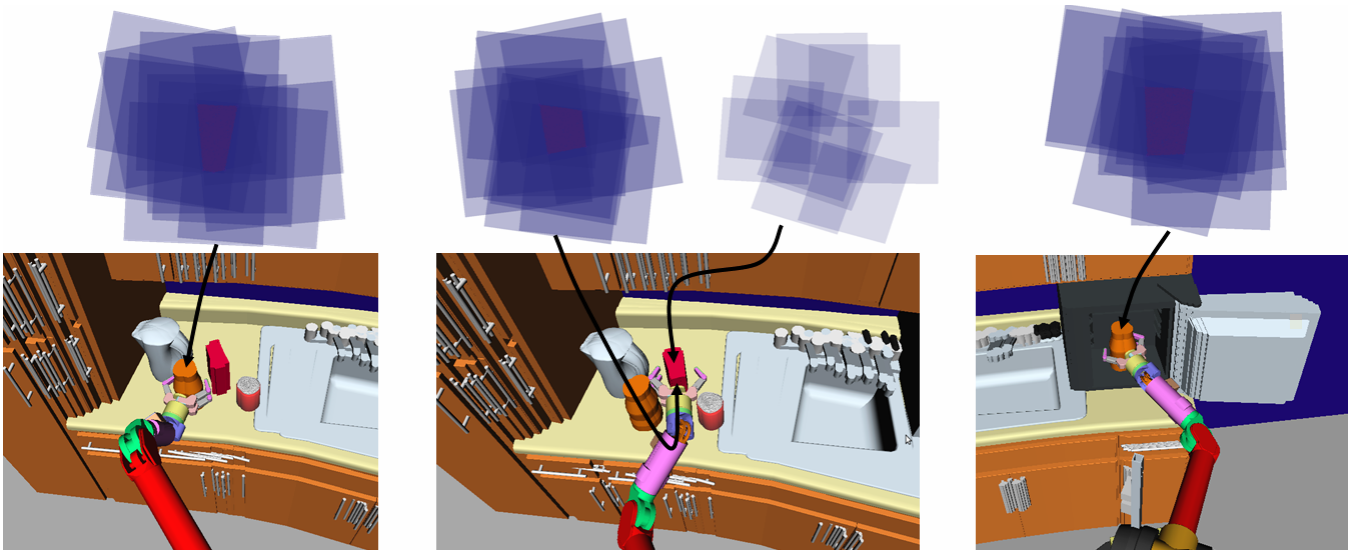

Fig. 7. Typical results of tasks 1, 2, and 3, from left to right. The intersecting boxes above show several of the intersecting TSRs for these tasks. In task 2 the TSR for grasping the box from the top is eliminated by the uncertainty (there is no point where all the boxes intersect) while the one for grasping it from the side is not.

	Number of Hypotheses				
Scene	1	15	30	45	60
1 (juice)	1.53 (100%)	3.48 (100%)	8.55 (100%)	12.29 (100%)	24.25 (100%)
1 (rice)	1.33 (100%)	6.37 (100%)	11.95 (100%)	14.64 (100%)	22.54 (100%)
2 (juice)	4.72 (100%)	23.73 (90%)	40.95 (80%)	39.56 (70%)	52.22 (50%)

Table II: Runtimes(in seconds) and Percent Success for Planning

VII. Conclusion

We have presented an approach to addressing pose uncertainty in manipulation planning problems. After describing a reaching or object placement task with a set of TSRs, we can modify this set to take into account pose uncertainty by intersecting copies of each TSR in 6D pose space. If there is no point were all copies of a TSR intersect, that TSR is discarded. If all TSRs for a task are discarded, then we know that the task cannot be accomplished given the current pose uncertainty. If any TSRs remain after intersection, we invoke the IKBiRRT planner which trades-off exploring the robot's C-space and sampling from the TSRs to generate a path. Finally, we presented several experiments on the WAM arm in a cluttered kitchen environment. We found that our approach was an efficient way to generate collision-free plans that are guaranteed to meet task specifications in the presence of pose uncertainty.

VIII. Acknowledgements

Dmitry Berenson was partially supported by Intel Research Pittsburgh and by the National Science Foundation under Grant No. EEC-0540865. Thanks to Nico Blodow for many helpful discussions.

References

[1] D. Berenson, S. Srinivasa, D. Ferguson, A. Collet, and J. Kuffner, "Manipulation planning with workspace goal regions," in *ICRA*, 2009.
[2] T. Lozano-Perez, M. Mason, and R. H. Taylor, "Automatic synthesis of fine-motion strategies for robots," *IJRR*, vol. 3, no. 1, 1984.
[3] M. Erdmann, "Using backprojections for fine motion planning with uncertainty," *IJRR*, vol. 5, no. 1, p. 19, 1986.
[4] J. Canny, "On computability of fine motion plans," in *ICRA*, 1989, pp. 177–182.
[5] S. LaValle and J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *WAFR*, 2000.
[6] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.
[7] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *IROS*, 2007.
[8] Y. Hirano, K. Kitahama, and S. Yoshizawa, "Image-based object recognition and dexterous hand/arm motion planning using rrts for grasping in cluttered scene," in *IROS*, 2005.
[9] D. Montana, "Contact stability for two-fingered grasps," in *ICRA*, 1992.
[10] H. Hanafusa and H. Asada, *Robot Motion*. The MIT Press, 1982, ch. Stable prehension by a robot hand with elastic fingers.
[11] M. R. Cutkosky, *Analysis for an Active Robot Hand*. Kluwer Academic Publishers, 1985.
[12] V.-D. Nguyen, "Constructing stable grasps," *IJRR*, 1989.
[13] J. Ponce, S. Sullivan, A. Sudsang, J.-D. Boissonnat, and J.-P. Merlet, "On computing four-nger equilibrium and force-closure grasps of polyhedral objects," *IJRR*, vol. 16, no. 1, pp. 11–35, 1997.
[14] N. Pollard, "Closure and quality equivalence for efficient synthesis of grasps from examples," *IJRR*, vol. 23, no. 6, pp. 595–613, 2004.
[15] R. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *IJRR*, vol. 5, no. 4, pp. 56–68, May 1986.
[16] M. Sallinen, "Modelling and estimation of spatial relationships in sensor-based robot workcells," Ph.D. dissertation, VTT Electronics/University of Oulu, Oulu, Finland, 2003.
[17] L. Devroye, *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, 1986.
[18] J. Kuffner and S. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *ICRA*, 2000, pp. 995–1001.
[19] P. Chen and Y. Hwang, "SANDROS: a dynamic graph search algorithm for motion planning," *Robotics and Automation, IEEE Transactions on*, vol. 14, no. 3, pp. 390–403, Jun 1998.
[20] L. Sciavicco and B. Siciliano, *Modeling and Control of Robot Manipulators*, 2nd ed. Springer, 2000, pp. 96–100.