Manipulation Planning with Workspace Goal Regions

Dmitry Berenson[†] Siddhartha S. Srinivasa[‡†] Dave Ferguson[‡†] Alvaro Collet[†] James J. Kuffner[†]

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

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

***Abstract*— We present an approach to path planning for manipulators that uses Workspace Goal Regions (WGRs) to specify goal end-effector poses. Instead of specifying a discrete set of goals in the manipulator's configuration space, we specify goals more intuitively as volumes in the manipulator's workspace. We show that WGRs provide a common framework for describing goal regions that are useful for grasping and manipulation. We also describe two randomized planning algorithms capable of planning with WGRs. The first is an extension of RRT-JT that interleaves exploration using a Rapidly-exploring Random Tree (RRT) with exploitation using Jacobian-based gradient descent toward WGR samples. The second is the IKBiRRT algorithm, which uses a forward-searching tree rooted at the start and a backward-searching tree that is seeded by WGR samples. We demonstrate both simulation and experimental results for a 7DOF WAM arm with a mobile base performing reaching and pick-and-place tasks. Our results show that planning with WGRs provides an intuitive and powerful method of specifying goals for a variety of tasks without sacrificing efficiency or desirable completeness properties.**

I. Introduction

Many practical manipulation tasks afford a large amount of freedom in the choice of grasps, arm configurations, 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. If robots are to perform the same tasks as humans, they must also be able to take advantage of the freedom afforded by such task requirements.

In this paper we introduce the concept of Workspace Goal Regions (WGRs), which allows the specification of continuous regions in the six-dimensional workspace of end-effector poses as goals for a planner. A given task can entail any number of WGRs, each of which encompasses a subspace of any dimension less than or equal to six.

In previous work [1, 2], researchers have tackled this problem by sampling some number of end-effector poses from the goal regions and using inverse kinematics(IK) to find joint configurations which place the end-effector at the sampled locations. These configurations are then set as goals for a randomized planner, such as an RRT or BiRRT [3, 4]. While often capable of solving the problem at hand, this approach is neither probabilistically complete nor efficient. The issue is that some number of samples from the goal regions are chosen a-priori as goal configurations, and the planner is forced to use only these goals. If the chosen goal configurations are unreachable, the planner will fail (even given infinite time) though it may have succeeded if some other samples had been drawn from the goal regions. Even if some of the goal configurations are reachable, it may be very difficult to find a path to those configurations, necessitating a long planning time. Again, if some other samples are drawn from the goal region, the planning time may be very short.

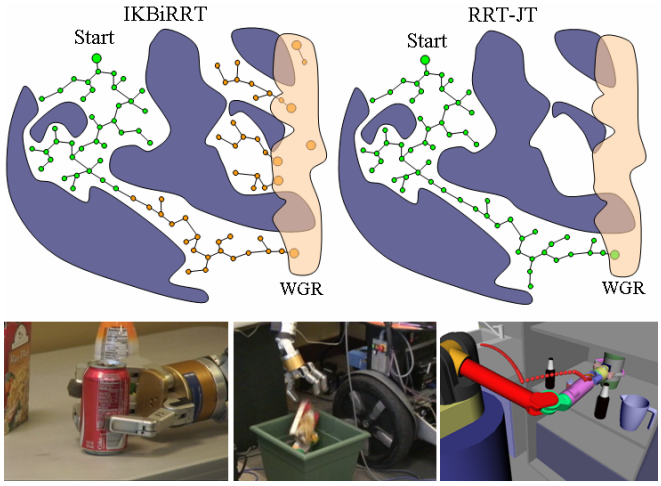


Fig. 1. Top: Depiction of the IKBiRRT and RRT-JT algorithms searching in c-space. The blue regions are obstacles, the forward-searching tree is shown with green nodes, and the backward-searching tree is shown with orange nodes. Bottom: Snapshots of the IKBiRRT running on the WAM arm in real and simulated environments. Left: Reaching to grasp a soda can. Center: Throwing away a box. Right: Trajectory for putting a bottle onto a cluttered refrigerator shelf.

Another approach to planning with certain types of workspace goals is to explore the configuration space (henceforth, c-space) of the robot with a single search tree that uses heuristics to bias the exploration toward a goal region [5]. However, the goal regions and heuristics defined in [5] are highly problem-specific and difficult to tune. Drumwright and Ng-Thow-Hing[6] employ a similar strategy of extending toward a randomly-generated IK solution for a workspace point. In [7], Vande Weghe *et al.* present the RRT-JT algorithm, which uses a forward-searching tree to explore the C-space and a gradient-descent heuristic based on the Jacobian-transpose to bias the tree toward a workspace goal point.

In this paper, we present two probabilistically complete planners: an extension of RRT-JT, and a new algorithm called IKBiRRT (see Figure 1). Both algorithms function by interleaving exploration of the robot's c-space with exploitation of WGRs. The IKBiRRT is intended for robots that have analytical or pseudo-analytical inverse-kinematics algorithms.

The extended RRT-JT is designed for robots that do not have such algorithms.

In the following sections we motivate and introduce WGRs and show how they can be used for randomized path planning. We describe each algorithm's implementation as well as our approach to inverse kinematics. We then demonstrate the algorithms on several example problems. To our knowledge these algorithms are the first probabilistically complete planners that handle a workspace goal representation as broad as WGRs.

II. Introducing Workspace Goal Regions

In general a set of workspace goals for a manipulator's end-effector can consist of an arbitrary number of six-dimensional points spread in an arbitrary way throughout the workspace. However, such a broad representation lacks three fundamental properties that are necessary for randomized planning.

First, the set of goal points 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 difficult and/or computationally expensive to discretize and input into a planner.

Second, sampling from the set of goal points must be efficient and 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, the distance to the goal set must be easily measurable when searching with a forward-only strategy. In order to bias the search toward the goal set, we must be able to compute a distance to that set from any point in the configuration space of the robot. Computing the shortest distance to any point in the set may be too time consuming if there are many points.

WGRs approach the problem of specifying goal sets by describing 6-dimensional volumes in the workspace. 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. WGRs are also intuitive to specify, can be efficiently sampled, and the distance to a WGR can be evaluated very quickly. A set of WGRs can describe any arbitrary set of goal points by, in the extreme case, assigning one WGR to every point, though this is clearly undesirable in practice because the specification, sampling, and distance checking 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 WGRs to specify our workspace goals.

WGRs 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. WGR Definition

Throughout this paper, we will be using transformation matrices of the form $\mathbf{T}_b^a$, 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 $\mathbf{t}_b^a$.

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

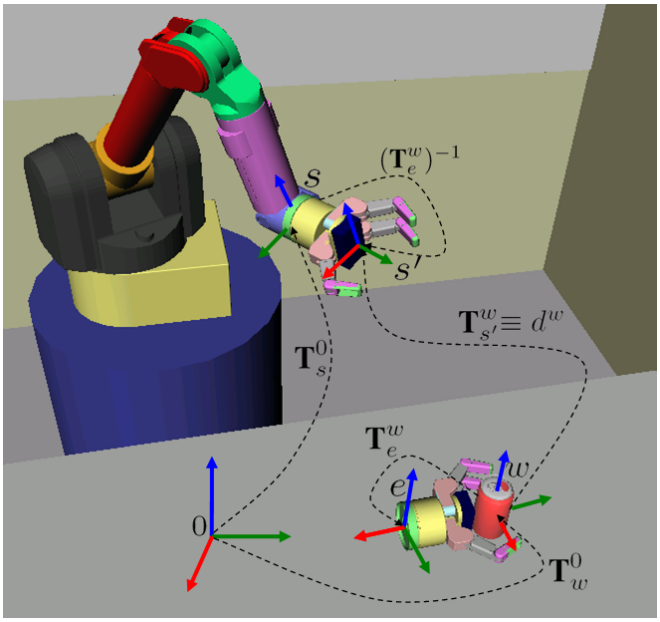


Fig. 2. Depiction of the transforms and coordinate frames involved in computing the distance to WGRs. The robot is in a sample configuration which has end-effector transform s and the hand near the soda can at transform e represents the end-effector offset transform defined by the WGR.

A WGR consists of three parts:

- $\mathbf{T}_w^0$: reference transform of the WGR 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 intuitively specified.

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 WGR 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 2 and Figure 4.

III. Using WGRs in Randomized Planning

In order to use WGRs in a randomized planner, we must be able to sample from a set of WGRs efficiently and, in the case of forward-only search, evaluate the distance of a configuration in the robot's c-space to the nearest WGR.

A. Distance to Workspace Goal Regions

In the RRT-JT algorithm, it will be necessary to find the distance from a given configuration q_s to a WGR (see Figure 2). Given a q_s, we use forward kinematics to get the position of the end-effector at this configuration $\mathbf{T}_s^0$. We then apply the inverse of the offset $\mathbf{T}_e^w$ to get $\mathbf{T}_{s'}^0$, which is the pose of the grasp location or the pose of the object held by the hand in world coordinates.

$$\mathbf{T}_{s'}^0 = \mathbf{T}_s^0 (\mathbf{T}_e^w)^{-1} \tag{3}$$

We then convert this pose from world coordinates to the coordinates of w.

$$\mathbf{T}_{s'}^w = (\mathbf{T}_w^0)^{-1} \mathbf{T}_{s'}^0 \tag{4}$$

Now we convert the transform $\mathbf{T}_{s'}^w$ into a 6×1 displacement vector from the origin of the w frame. This displacement represents rotation in the RPY convention so it is consistent with the definition of $\mathbf{B}^w$.

$$d^w = \begin{bmatrix} \mathbf{t}_{s'}^w \\ \arctan 2(\mathbf{R}_{s'_{32}}^w, \mathbf{R}_{s'_{33}}^w) \\ -\arcsin(\mathbf{R}_{s'_{31}}^w) \\ \arctan 2(\mathbf{R}_{s'_{21}}^w, \mathbf{R}_{s'_{11}}^w) \end{bmatrix} \tag{5}$$

Taking into account the bounds of $\mathbf{B}^w$, we get the 6×1 displacement vector to the WGR $\Delta\mathbf{x}$

$$\Delta\mathbf{x}_i = \begin{cases} d_i^w - \mathbf{B}_{i,1}^w & \text{if } d_i^w < \mathbf{B}_{i,1}^w \\ d_i^w - \mathbf{B}_{i,2}^w & \text{if } d_i^w > \mathbf{B}_{i,2}^w \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where i indexes through the six rows of $\mathbf{B}^w$ and six elements of $\Delta\mathbf{x}$ and d^w. $\|\Delta\mathbf{x}\|$ is the distance to the WGR. Note that we implicitly weight rotation in radians and translation in meters equally when computing $\|\Delta\mathbf{x}\|$ but the two types of units can be weighted in an arbitrary way to produce a distance metric that considers one or the other more important.

B. Sampling From Workspace Goal Regions

Both the IKBiRRT and the RRT-JT require that samples be drawn from WGRs so that they can be used for inverse kinematics or gradient-descent. Sampling from a single WGR 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_{sample}^w and converted into the transformation $\mathbf{T}_{sample}^w$. We can then convert this sample into world coordinates after applying the end-effector transformation.

Algorithm 1: IKBiRRT(q_s, W)

```
T_a.Init(q_s); T_b.Init(NULL);
while TimeRemaining() do
    T_goal = GetBackwardTree(T_a, T_b);
    if T_goal.size = 0 or rand(0,1) < P_sample then
        AddIKSolutions(T_goal, W);
    else
        q_rand ← RandConfig();
        q^a_near ← NearestNeighbor(T_a, q_rand);
        q^a_reached ← Extend(T_a, q^a_near, q_rand);
        q^b_near ← NearestNeighbor(T_b, q^a_reached);
        q^b_reached ← Extend(T_b, q^b_near, q^a_reached);
        if q^a_reached = q^b_reached then
            P ← ExtractPath(T_a, q^a_reached, T_b, q^b_reached);
            return SmoothPath(P);
        else
            Swap(T_a, T_b);
        end
    end
end
return NULL;
```

$$\mathbf{T}_{sample'}^0 = \mathbf{T}_w^0 \mathbf{T}_{sample}^w \mathbf{T}_e^w \tag{7}$$

However, in the case of multiple WGRs, we must first decide which WGR to sample from. If all WGRs enclose six-dimensional volumes, we can choose among WGRs in proportion to their volume. However a volume-proportional sampling will ignore WGRs 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 WGRs proportional to the *sum* of the differences between their bounds.

$$\zeta_i = \sum_{j=1}^{6} \left(\mathbf{B}_{j,2}^{w_i} - \mathbf{B}_{j,1}^{w_i} \right) \tag{8}$$

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

IV. The IKBiRRT Algorithm

The IKBiRRT (see Algorithm 1) is an extension of the Bidirectional RRT (BiRRT) algorithm 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 WGRs W. 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[8]; 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 point in W using the method described in section III-B, 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 possible configuration in the WGRs is added to the backwards search tree. This is because, as time goes to infinity, every point in these workspace regions will be sampled, and have a set of its IK solutions added to the tree an infinite number of times. As long as the IK solver used is also probabilistically complete (i.e. will return all possible IK solutions as the number of times it is invoked approaches infinity), this guarantees that all possible goal configurations will be added to the backwards search tree. Since the BiRRT algorithm is also probabilistically complete, as time goes to infinity the forwards and backwards trees will thus connect and return a solution involving one of these goal configurations.

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 Jacobian pseudo-inverse or Jacobian-transpose methods[9] 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.

Algorithm 2: Extended RRT-JT(q_s, W)

```
1  T.Init(q_s);
2  while TimeRemaining() do
3      if rand(0,1) < P_sample then
4          q_sample ← WeightedSampleNode(T);
5          W_i ← DistWeightedSampleWGR(q_sample, W);
6          T^0_sample' ← SampleWGR(W_i);
7          Q_new ← GradientExtend(q_sample, T^0_sample');
8      else
9          q_rand ← RandConfig();
10         q_near ← NearestNeighbor(T, q_rand);
11         Q_new ← Extend( q_near, q_rand);
12     end
13     D ← DistanceToNearestWGR(Q_new, W);
14     T.AddNodes(Q_new, D);
15     for i = 1,2,...D.size do
16         if D_i = 0 then
17             P ← T.extractPath(Q_new_i);
18             return SmoothPath(P);
19         end
20     end
21 end
22 return NULL;
```

Algorithm 3: GradientExtend(q_{sample}, $T^0_{sample'}$)

```
1  Q_new ← {};
2  while true do
3      q_s^old ← q_s;
4      T^0_s ← ForwardKinematics(q_sample);
5      Δx ← TransformDifference(T^0_s, T^0_sample');
6      T^0_step ← GetIntermediateTransform(T^0_s, T^0_sample');
7      q_s ← GradientDescent(q_s, T^0_step);
8      if q_s ≠ NULL and CollisionFree(q_s^old, q_s) then
9          Q_new ← Q_new ∪ q_s;
10     else
11         return Q_new;
12     end
13     if ||Δx|| = 0 then
14         return Q_new;
15 end
```

In the case of a mobile base, our IK solver first samples a base position within a circle of the given end-effector transform. The radius of the circle is the length of the arm when it is fully outstretched. Once the base position is sampled, the above pseudo-analytical inverse kinematics solver is called. 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 W.

V. The Extended RRT-JT Algorithm

The extended RRT-JT (see Algorithm 2) works by using similar principles to the IKBiRRT but is intended for manipulators where an analytical or pseudo-analytical IK solution is not available or too time consuming. Unlike the IKBiRRT which grows two trees, RRT-JT grows a single tree in the c-space, which alternates between exploring the c-space and gradient-descending toward a sample from W. Again, P_{sample} controls how often the gradient descent step is executed in proportion to the exploration step.

When not gradient-descending, RRT-JT uses the same Extend function as the IKBiRRT except that this function returns the nodes along the path toward q_{rand} (Q_{new}) instead of adding them to the tree within the function. Scores (D) are assigned to these nodes by measuring the distance squared to each WGR in W and taking the smallest. The distance is measured using the process described in section III-A. If the distance from any node to a WGR is 0, the algorithm has found a path to a goal configuration and this path is smoothed and returned as with the IKBiRRT.

When the gradient-descent step is chosen, the WeightedSampleNode function samples a random node from T with probability inversely proportional to its score, thus favoring nodes that are closer to WGRs. The DistWeightedSampleWGR function then samples a WGR from W with probability inversely proportional to that WGR's squared distance from q_{sample}. Finally, the SampleWGR function samples a transform from W_i using the methods in III-B to generate $\mathbf{T}^0_{sample'}$.

The GradientExtend function (see Algorithm 3) works by iteratively generating intermediate transformations between the end-effector position specified by executing forward kinematics on q_{sample} and $T^0_{sample'}$. For each intermediate transformation, we run an iterative Jacobian pseudo-inverse gradient descent[9] to move the end-effector to that transform (see Algorithm 4). This process terminates when the gradient descent fails because it reaches a joint limit, a collision occurs, or the distance to $\mathbf{T}^0_{sample'}$ is less than ϵ. Again, the distance from nodes generated by GradientExtend (Q_{new}) to the nearest WGR is computed and if that distance is 0, the algorithm has found a path.

The extended RRT-JT algorithm is also probabilistically complete. This follows because, by randomly sampling configurations and extending the tree toward these configurations some fraction of the time (lines 9 through 11 of Algorithm 2), the extended RRT-JT algorithm retains the probabilistic completeness property inherent in the original RRT algorithm.

VI. Results

To illustrate the effectiveness and compare the performance of the proposed algorithms, we tested them on four example problems in simulation as well as on our robot. We consider both the fixed-base case of the robot, where we plan for the 7DOF of the arm, as well as the mobile-base case where we plan for the 7DOF of the arm and 2DOF of translation, i.e. in 9DOF. No non-holonomic constraints are placed on the base, though such constraints could be incorporated into the planner,

Algorithm 4: GradientDescent(q_s, $\mathbf{T}^0_{step}$)

```
1 while true do
2     T^0_s ← ForwardKinematics(q_s);
3     Δx ← TransformDifference(T^0_step, T^0_s);
4     if ‖Δx‖ < ε then return q_s;
5     J ← GetJacobian(q_s);
6     Δq_error ← J^T (JJ^T)^{-1} Δx;
7     q_s ← (q_s − Δq_error);
8     if OutsideJointLimit(q_s) then return NULL;
9 end
```

as in [10]. The four problems and examples of trajectories found by the algorithms are shown in Figure 3.

A. Problem 1: Reaching to Grasp an Object

Our goal in this problem is to grasp an object for which we can define a continuum of acceptable grasp poses. These grasp poses can be encoded into WGRs and passed to either planner. We define four WGRs for the pitcher we wish to grasp (see Figure 3(a)): two for the top of the pitcher and two for the handle. The $\mathbf{T}^0_w$ and $\mathbf{T}^w_e$ transforms of these WGRs are shown in Figure 4. The two bounds for the top WGRs are identical, as are the two bounds for the handle WGRs.

$$\mathbf{B}^w_{top} = \begin{bmatrix} \mathbf{0}_{5\times 2} \\ -0.3 \quad 0.3 \end{bmatrix} \mathbf{B}^w_{handle} = \begin{bmatrix} \mathbf{0}_{2\times 2} \\ -0.03 \quad 0.02 \\ \mathbf{0}_{3\times 2} \end{bmatrix} \quad (9)$$

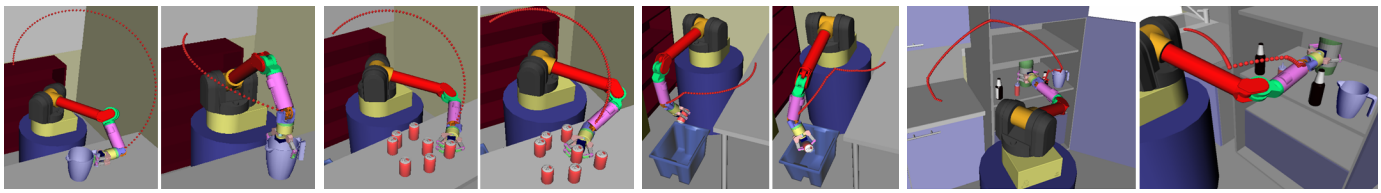
The top WGRs allow the robot to grasp the pitcher from the top with limited hand rotation about the z-axis. The handle WGRs allow the robot to grasp the pitcher anywhere along the handle but do not allow any offset in hand rotation. Trajectories produced by our planners are shown in Figure 3(a).

B. Problem 2: Reaching to Grasp Multiple Objects

In this problem the robot's task is to reach and grasp one of seven randomly-placed soda cans on a table (see Figure 3(b)). Each soda can is treated as a cylinder and two WGRs are defined for each can. The $\mathbf{T}^0_w$ and $\mathbf{T}^w_e$ transforms are shown in Figure 4. Both WGRs for each can have identical bounds:

$$\mathbf{B}^w = \begin{bmatrix} \mathbf{0}_{5\times 2} \\ -\pi \quad \pi \end{bmatrix} \quad (10)$$

These bounds allow the grasp to rotate about the z-axis of the can, thus allowing it to grasp the can from any direction in the plane defined by the x and y coordinates of the can's center. Note that we do not specify which soda can to grasp, this choice is made within the planner when sampling from the WGRs. Trajectories produced by our planners are shown in Figure 3(b).



(a) Problem 1 (b) Problem 2 (c) Problem 3 (d) Problem 4

Fig. 3. Trajectories of the end-effector produced by the planners for the four simulation problems. (a)-(c) Reaching to grasp a pitcher, reaching to grasp a soda can, and throwing away a soda can, left: IKBiRRT trajectory, right: RRT-JT trajectory. (d) Placing a bottle into the refrigerator using IKBiRRT, left: fixed base, right: mobile base. The trajectories shown have been smoothed with 500 iterations of the shortcut smoothing algorithm.

C. Problem 3: Throwing Away an Object

The goal of this problem is to show the planners' performance on tasks where there is a large WGR. The robot must place the soda can it is holding into a region above the recycling bin and then release its grasp (see Figure 3(c)). $\mathbf{T}_w^0$ is defined at the center of the upper rim of the recycling bin and $\mathbf{T}_e^w$ is defined as an end-effector position pointing along x (away from the robot) that is holding the can at $\mathbf{T}_w^0$.

$$\mathbf{B}^w = \begin{bmatrix} -0.15 & 0.15 \\ -0.1 & 0.1 \\ -0.03 & 0.03 \\ -\pi & \pi \\ -\pi & \pi \\ -\pi & \pi \end{bmatrix} \tag{11}$$

This $\mathbf{B}^w$ defines a box above the recycling bin where all rotations of the object are allowed.

D. Problem 4: Placing an Object into a Constrained Space

This problem differs from the previous one conceptually in that the object held by the robot must be placed into a very cluttered location (see Figure 3(d)). The bottles in the refrigerator and the upper refrigerator shelf make it difficult for the robot to find a path that places the large bottle it is holding onto the middle refrigerator shelf. $\mathbf{T}_w^0$ is defined at the center of the middle shelf and $\mathbf{T}_e^w$ is defined as an end-effector position pointing along y (away from the robot) that is holding the bottle at $\mathbf{T}_w^0$.

$$\mathbf{B}^w = \begin{bmatrix} -0.24 & 0.24 \\ -0.34 & 0.34 \\ \mathbf{0}_{4\times 2} \end{bmatrix} \tag{12}$$

This $\mathbf{B}^w$ defines a plane on the shelf where the bottle can be placed.

E. Runtimes and Parameter-Tuning

Both the IKBiRRT and the RRT-JT rely on the P_{sample} parameter to determine how often to sample from or gradient-descend toward a WGR. To determine an acceptable value for this parameter, we iterate over ten values, $P_{sample} = 0.05, 0.15, ..., 0.95$. For each value of P_{sample} and each problem, we run both algorithms 15 times for the fixed-base robot and another 15 times for the mobile-base robot. The resulting runtimes are shown in Figure 5. These runtimes do not include smoothing because it is limited to a constant time defined by the user.

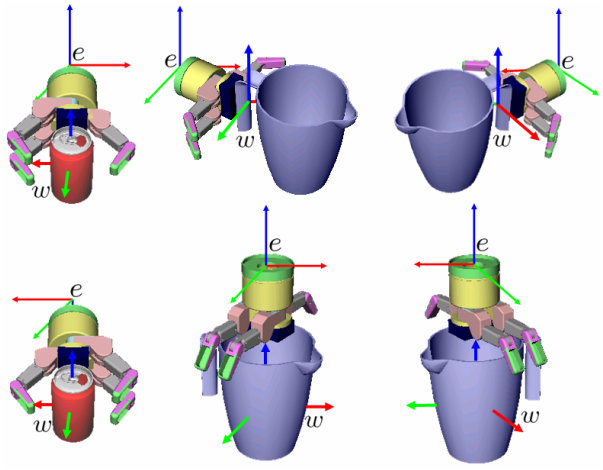
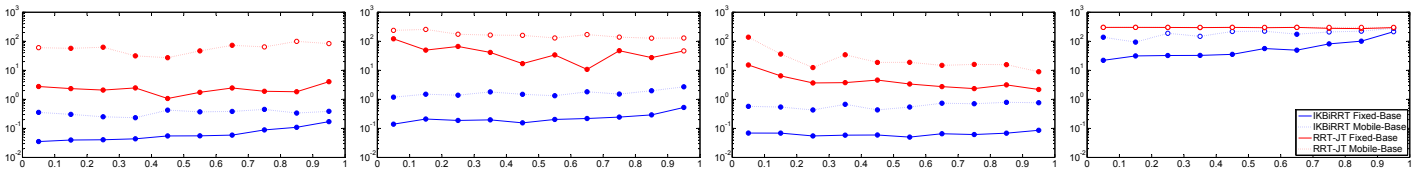


Fig. 4. Depiction of the w and e frames which are used to get the $\mathbf{T}_w^0$ and $\mathbf{T}_e^w$ transforms for the soda can and pitcher when reaching to grasp.

From the data in Figure 5 we can surmise that, in general, P_{sample} for the IKBiRRT algorithm should be somewhere between 0.05 and 0.45. P_{sample} in this range yielded values close to the minimum run times for each problem. The best P_{sample} values for the RRT-JT were in the 0.45 to 0.65 range.

It is important to note that the RRT-JT was not able to reliably solve the fixed-base and the mobile-base cases of Problem 4 for any P_{sample} value. This is because almost all of the straight-line gradient-descents chosen by RRT-JT toward the shelf result in collision with a bottle on the shelf or with the upper shelf. This highlights the usefulness of a backward tree (like in the IKBiRRT) for escaping from a constrained goal region.

Overall, the IKBiRRT clearly outperformed RRT-JT on the four simulation problems with average runtimes of 0.035s, 0.140s, 0.050s, and 22.1s for the best P_{sample} values for Problems 1-4 (fixed base). The RRT-JT achieved average runtimes of 1.07s, 10.8s, 2.19s, and 273.4s for the best P_{sample} values for Problems 1-4 (fixed base). This gap in performance is explained by the fact that the IKBiRRT uses an IK solver to create roots for the backward searching tree. Since the IK-solver is very fast, creating the backward tree can be done quickly and it is well known that searching bidirectionally



(a) Problem 1 (b) Problem 2 (c) Problem 3 (d) Problem 4

Fig. 5. Average runtimes plotted on a log scale for ten values of P_{sample}. Solid data points represent tests where all 15 runs took less than 5 minutes each, hollow data points represent tests where at least one run exceeded the 5 minute time-limit and was terminated. A terminated run counted as a run of 5 minutes when computing the averages. The legend in the right-most graph applies to all graphs.

will tend to be faster than a single-tree approach. The one advantage the RRT-JT possess is that it does not require any kind of IK solver and thus can be applied to problems for robots with large numbers of degrees of freedom where computing a pseudo-analytical solution to the IK is too inefficient.

F. Experiments on Robot

To demonstrate the effectiveness of the IKBiRRT on our robot, we ran several experiments on our physical WAM arm. The arm's task was to reach and grasp a series of objects placed in a cluttered arrangement on a table and throw them in a trash bin. Conceptually, this task is a combination of the tasks in problems 2 and 3.

The 6D poses of the objects on the table were determined using a camera mounted on the robot. The algorithm for determining their poses is described in [11]. Each object is approximated as either a box or a cylinder for the purposes of determining the grasp pose and multiple WGRs are assigned to each object as with the pitcher and soda cans in problems 1 and 2. No order is specified for grasping the objects, all WGRs for all objects are input into the IKBiRRT, which returns the first valid trajectory found. Several snapshots of the execution of this task are shown in Figure 6. Please see our video for several runs of this experiment:

http://www.cs.cmu.edu/%7edberenso/wgrplanning.mp4

Fig. 6. Snapshots from three runs of the IKBiRRT planner in cluttered scenes. Top Row: Grasping and throwing a way a box of rice. Middle Row: Grasping and throwing away a juice bottle. Bottom Row: Grasping and throwing away a soda can.

VII. Conclusion

We have presented planning with Workspace Goal Regions as an intuitive yet powerful method for manipulator path planning. We have also presented two probabilistically-complete algorithms, the IKBiRRT and an extended version of the RRT-JT, that use WGRs to plan c-space trajectories for a variety of tasks. Both planners interleave exploring the c-space and exploiting the WGRs by using only one parameter P_{sample}. The planners were demonstrated on four example problems in simulation and their results were compared in terms of runtimes. The IKBiRRT outperformed the RRT-JT on all example problems because of its ability to search bidirectionally and its use of a fast IK solver, however the RRT-JT can be used when there is no such solver available. The IKBiRRT was also implemented on a 7DOF WAM arm, where it was used to perform a typical clean-up task.

VIII. Acknowledgements

Dmitry Berenson and Alvaro Collet were partially supported by the Intel Summer Fellowship awarded by Intel Research Pittsburgh and by the National Science Foundation under Grant No. EEC-0540865.

References

[1] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *IROS*, 2007.

[2] 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.

[3] S. LaValle and J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *WAFR*, 2000.

[4] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

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

[6] E. Drumwright and V. Ng-Thow-Hing, "Toward interactive reaching in static environments for humanoid robots," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 846–851.

[7] M. Vande Weghe, D. Ferguson, and S. Srinivasa, "Randomized path planning for redundant manipulators without inverse kinematics," in *Humanoids*, 2007.

[8] 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.

[9] L. Sciavicco and B. Siciliano, *Modeling and Control of Robot Manipulators*, 2nd ed. Springer, 2000, pp. 96–100.

[10] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," *Internation Journal of Robotics Research*, vol. 20, Jun 2001.

[11] A. Collet, D. Berenson, S. Srinivasa, and D. Ferguson, "Object recognition and full pose registration from a single image for robotic manipulation," in *ICRA*, 2009.