Pose-Constrained Whole-Body Planning using Task Space Region Chains

Dmitry Berenson[1] Joel Chestnutt[2] Siddhartha S. Srinivasa[1,3] James J. Kuffner[1,2] Satoshi Kagami[2]

[1]*The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 15213, USA* [dberenso, kuffner]@cs.cmu.edu
[2]*Digital Human Research Center, AIST, Tokyo, Japan* [joel.chestnutt, s.kagami]@aist.go.jp
[3]*Intel Research Pittsburgh, Pittsburgh, PA, 15213, USA* siddhartha.srinivasa@intel.com

***Abstract*— We present an efficient approach to generating paths for humanoids and other robotic manipulators that uses the Task Space Region (TSR) framework to specify manipulation tasks. TSRs can define acceptable goal poses of an end-effector or constraints on the end-effector's pose during the path, or both. First presented as a method for goal-specification [1], TSRs are a straightforward representation of sets of end-effector poses which can be sampled and which entail a clear distance metric. This makes TSRs ideal for sampling-based motion planning. However, a finite set of TSRs is sometimes insufficient to capture the pose constraints of a given task. To describe more complex constraints, we present TSR Chains, which are defined by linking a series of TSRs. Though the sampling for TSR Chains follows clearly from that of TSRs, the distance metric for TSR Chains is radically different. We also present a new version of our Constrained Bidirectional RRT (CBiRRT2) planner, which is capable of planning with TSR chains as well as other constraints. We demonstrate our approach on the HRP3 robot by performing a variety of whole-body manipulation tasks.**

I. Introduction

Many practical manipulation tasks, like moving a large box or opening a refrigerator door, impose constraints on the motion of a robot's end-effector. Many such tasks also afford significant freedom in the acceptable goal pose of the end-effector. For example in Figure 1, although the humanoid HRP3's hands are constained to grasp the box during manipulation, the task of placing the box on the table affords a wide range of box placements and robot configurations that achieve the goal. We propose a novel framework for pose-constrained manipulation planning which is capable of satisfying constraints and affordances to produce manipulation plans for high DOF robots, like humanoids, in a few seconds.

Our framework comprises of two parts: a novel unifying representation of constraints and affordances which we term Task Space Regions (TSRs), and a sampling-based planner, the CBiRRT2, that exploits the unique properties of TSRs. Building on prior work [1][2], TSRs describe goal and constraint sets as volumes in $SE(3)$, the space of rigid spatial transformations. We showed that such volumes are particularly useful for specifying manipulation tasks such as reaching to grasp an object, or manipulating objects with pose constraints.

While we showed that TSRs are intuitive to specify, can be quickly sampled, and the distance to TSRs can be evaluated efficiently [1], a single TSR, or even a finite set of TSRs, is sometimes insufficient to capture the pose constraints of a given task. To describe more complex constraints such as closed chain kinematics and manipulating articulated objects, this paper introduces the concept of TSR Chains, which are defined by linking a series of TSRs (Section IV). Though the sampling for TSR Chains follows clearly from that of TSRs, the distance metric for TSR Chains is radically different.

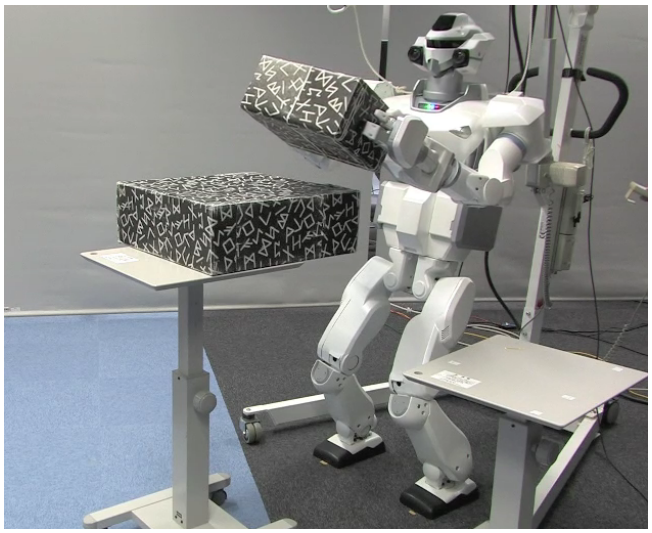


Fig. 1. HRP3 performing a box stacking task. The closed-chain kinematics of the feet and the hands constrain the poses of the robot's end-effectors.

We also present the CBiRRT2 planner, which is partly a combination of the IKBiRRT [1] and CBiRRT [2] planners that works with TSR Chains among other constraints (Section V). CBiRRT2 samples end-effectors goal poses to discover goal configurations of the robot and satisfies end-effector pose constraints through a process we call *IK handshaking*.

We demonstrate the efficiency and versatility of this planner and the TSR framework through planning for a diverse set of manipulation tasks for 28DOF of the HRP3 (Section VI).

The contributions of this paper are:

- A unified framework for specifying end-effector goals and pose constraints
- TSR Chains for specifying complex pose goals and constraints
- CBiRRT2 planner for planning with TSR Chains and other constraints

II. Background

A key feature of Task Space Regions is that they can be sampled to produce goal configurations for a sampling-based planner. Other researchers have approached the problem of ambiguous goal specification by sampling some number of configuration space (C-space) goals before running the planner [3][4], which limits the planner to a small set of solutions from a region that is really continuous. Another approach is to bias a single-tree planner toward the goal regions, however this approach usually considers only single points [5][6] in the task space or is hand-tuned for specialized goal regions [7].

The CBiRRT2 algorithm is partly a fusion of previously-proposed algorithms [1][2], which are based on the RRT [8]. It uses iterative inverse-kinematics techniques [9][10] to meet pose constraints and sample goal configurations. The algorithm plans in the C-space, which implicitly allows it to search the null-space of pose constraints, unlike task-space planners [11][12][13], which assign a single configuration to each task-space point (from a potentially infinite number of possible configurations). Exploration of the null-space can be useful for satisfying other constraints, such as avoiding obstacles or maintaining balance, however our constraint representation could be incorporated into task-space planners as well. Stilman [14] and Yakey et al. [15] proposed single-tree RRT planners that use various projection methods to meet certain types of pose constraints. Our approach is similar to Stilman's, however we differ in the planning method and provide a more general constraint representation, which is central to this paper.

III. Task Space Regions

This section briefly reviews the definition of TSRs. Sampling and distance checking methods for single TSRs can be found in our previous work [1].

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
- $\mathbf{T}_e^w$: offset transform in the coordinates of w
- $\mathbf{B}^w$: 6×2 matrix of bounds in the coordinates of w

In previous work [1] the 0 frame corresponded to the world origin, thus defining the TSR relative to that frame. However, in this paper we show how defining a TSR relative to a link of the robot can be useful for representing constraints such as closed-chain kinematics (see Section VI-B).

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

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. We use the Roll-Pitch-Yaw (RPY) Euler angle convention, which is employed because it allows bounds on rotation to be intuitively specified.

IV. Task Space Region Chains

Though TSRs can represent a broad range of useful pose constraints and goal poses, some constraints and goals cannot be well-described by a finite set of TSRs. For instance, consider the task of opening a door while allowing the end-effector to rotate about the door handle. It is straightforward to specify the rotation of the door about its hinge as a single TSR and to specify the rotation of the end-effector about the door's handle as a single TSR if the door's position is fixed. However, the product of these two constraints (allowing the end-effector to rotate about the handle while the door is moving) cannot be completely specified with a finite set of TSRs. In order to allow more complex constraint representations in the TSR framework, we present TSR Chains, which are constructed by linking a series of TSRs.

A. TSR Chain Definition

A TSR chain $\mathbf{C}$ consists of a set of n TSRs: $\mathbf{C} = \{\mathrm{TSR}_1, \mathrm{TSR}_2, ...\mathrm{TSR}_n\}$. The TSRs in $\mathbf{C}$ have the property

$$\mathbf{C}_i.\mathbf{T}_w^0 = (\mathbf{C}_{i-1}.\mathbf{T}_w^0)(\mathbf{C}_{i-1}.\mathbf{T}_{sample}^w)(\mathbf{C}_{i-1}.\mathbf{T}_e^w) \tag{3}$$

for $i = \{2...n\}$ where $\mathbf{C}_i$ corresponds to the ith TSR in the chain and $\mathbf{C}_i.\{\cdot\}$ refers to an element of the ith TSR. Of course a TSR Chain can consist of only one TSR, in which case it is identical to a normal TSR. $\mathbf{C}_i.\mathbf{T}_{sample}^w$ can be any transform obtained by sampling from inside the bounds of $\mathbf{C}_i.\mathbf{B}^w$. Thus we do not know $\mathbf{C}_i.\mathbf{T}_w^0$ until we have determined $\mathbf{T}_{sample}^w$ values for all previous TSRs in the chain.

In this way, a TSR chain can be thought of as a virtual serial-chain manipulator. Again consider the door example. To define the TSR chain for this example, we can imagine a virtual manipulator that is rooted at the door's hinge. The first link of the manipulator rotates about the hinge and extends from the hinge to the handle. At the handle, we define another link that rotates about the handle and extends to where a robot's end-effector would be if the robot were grasping the handle (see Figure 2). $\mathbf{C}_1.\mathbf{T}_{sample}^w$ would be a rotation about the door's hinge corresponding to how much the door had been opened. In this way the $\mathbf{T}_{sample}^w$ values for each TSR are analogous to transforms induced by the "joint angles" of the virtual manipulator. The joint limits of these virtual joints are defined by the values in $\mathbf{B}^w$.

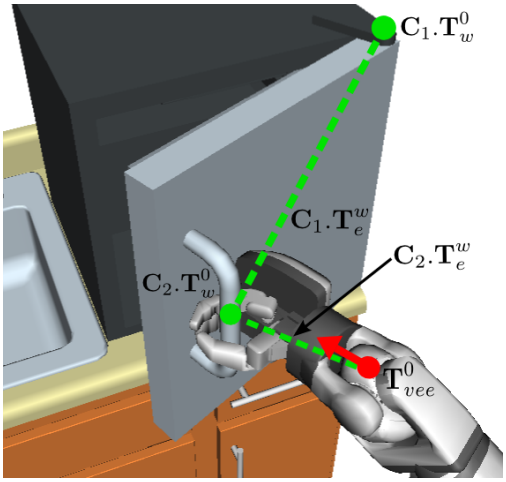


Fig. 2. Depiction of the virtual manipulator for the door example. The red dot and arrow represent the virtual end-effector which is at transform $\mathbf{T}_{vee}^0$.

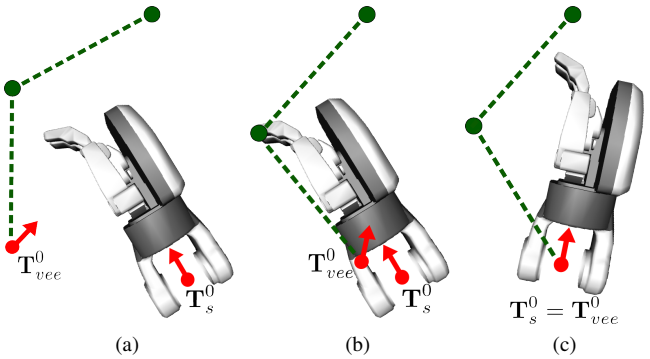


Fig. 3. Depiction of the IK handshaking procedure. (a) The virtual manipulator starts in some configuration. (b) Finding the closest configuration of the virtual manipulator. (c) The robot's manipulator moves to meet the constraint.

B. Sampling From TSR Chains

To determine root configurations for a bi-directional sampling-based planner, we will need to draw samples from a given TSR Chain. To do this, we first sample from within $\mathbf{C}_1.\mathbf{B}^w$ to obtain $\mathbf{C}_1.\mathbf{T}_{sample}^w$. This is done by sampling uniformly between the bounds in $\mathbf{B}^w$, compiling the sampled values into a displacement $d_{sample}^w = [x\ y\ z\ \psi\ \theta\ \phi]$ and converting that displacement into the transform $\mathbf{C}_1.\mathbf{T}_{sample}^w$. We then use this sample to determine $\mathbf{C}_2.\mathbf{T}_w^0$ via Equation 3. We repeat this process for each TSR in the chain until we reach the nth TSR. We then obtain a sample in the 0 frame:

$$\mathbf{T}_{sample'}^0 = (\mathbf{C}_n.\mathbf{T}_w^0)(\mathbf{C}_n.\mathbf{T}_{sample}^w)(\mathbf{C}_n.\mathbf{T}_e^w) \tag{4}$$

Note that the sampling of TSR chains in this way is biased but the sampling will cover the entire set. If there is more than one TSR Chain defined for a single manipulator, this means that we have the option of drawing a sample from any of these TSR Chains. We choose a TSR Chain for sampling with probability proportional to the sum of the differences between the bounds of all TSRs in that chain.

C. Distance to TSR Chains

Though the sampling method for TSR Chains follows directly from the sampling method for TSRs, evaluating distance to a TSR Chain is far different from evaluating distance to a TSR. This is because we do not know which $\mathbf{T}_{sample}^w$ values for each TSR in the chain yield the minimum distance to a query transform $\mathbf{T}_s^0$, which is derived from a query configuration q_s using forward kinematics.

To approach this problem, it is again useful to think of the TSR chain as a virtual manipulator (See Figure 3a). Finding the correct $\mathbf{T}_{sample}^w$ values for each TSR is equivalent to finding the joint angles of the virtual manipulator that bring its virtual end-effector as close to $\mathbf{T}_s^0$ as possible. Thus we can see this distance-checking problem as a form of the standard Inverse Kinematics (IK) problem, which is to find the set of joint angles that brings an end-effector to a given transform. Depending on the TSR Chain definition and $\mathbf{T}_s^0$, the virtual manipulator may not be able to reach the desired transform, in which case we want the virtual end-effector to get as close as possible. Thus we can apply standard iterative IK techniques based on the Jacobian pseudo-inverse [9] to move the virtual end-effector to a transform that is as close as possible to $\mathbf{T}_s^0$ (see Figure 3b). Once we obtain the joint angles of the virtual manipulator, we convert them to $\mathbf{T}_{sample}^w$ values and forward-chain to obtain the virtual end-effector position $\mathbf{T}_{vee}^0$. We then compute $\mathbf{T}_s^0$ in the virtual end-effector's frame:

$$\mathbf{T}_s^{vee} = (\mathbf{T}_{vee}^0)^{-1}\mathbf{T}_s^0 \tag{5}$$

and then convert to the displacement form:

$$d_s^{vee} = \begin{bmatrix} \mathbf{t}_s^{vee} \\ \arctan 2(\mathbf{R}_{s32}^{vee}, \mathbf{R}_{s33}^{vee}) \\ -\arcsin(\mathbf{R}_{s31}^{vee}) \\ \arctan 2(\mathbf{R}_{s21}^{vee}, \mathbf{R}_{s11}^{vee}) \end{bmatrix} \tag{6}$$

$\|d_s^{vee}\|$ is the distance between $\mathbf{T}_s^0$ and $\mathbf{T}_{vee}^0$. Note the equal weighting of rotation in radians and translation in meters.

Once the distance is evaluated, our planner calls the iterative IK algorithm for the robot's manipulator to move the robot's end-effector to $\mathbf{T}_{vee}^0$ to meet the constraint specified by this TSR Chain. We term this process of calling IK for the virtual manipulator and the robot in sequence *IK handshaking*.

Just as with TSR Chains used for sampling, we may define more than one TSR Chain as a constraint for a single manipulator. This means that we have the option of satisfying any of these TSR Chains to produce a valid configuration. To find which chain to satisfy, we perform the distance check from our current configuration to each chain and choose the one with the smallest distance.

D. Physical Constraints

In the door example, the first TSR corresponds to a physical joint of a body in the environment but the second one is purely virtual; i.e. defining a relation between two frames that is not enforced by a joint in the environment (in this case the relation is between the robot's end-effector and the handle of the door).

Algorithm 1: CBiRRT2(Q_s, Q_g)

T_a.Init(Q_s); T_b.Init(Q_g);
while TimeRemaining() **do**
  T_{goal} = GetBackwardTree(T_a, T_b);
  if T_{goal}.size $= 0$ ***or*** rand$(0,1) < P_{sample}$ **then**
    AddRoot(T_{goal});
  else
    $q_{rand} \leftarrow$ RandomConfig();
    $q^a_{near} \leftarrow$ NearestNeighbor(T_a, q_{rand});
    $q^a_{reach} \leftarrow$ **ConstrainedExtend**(T_a, q^a_{near}, q_{rand});
    $q^b_{near} \leftarrow$ NearestNeighbor(T_b, $q^a_{reached}$);
    $q^b_{reach} \leftarrow$ **ConstrainedExtend**(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
end
return NULL;

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

$q_s \leftarrow q_{near}$; $q^{old}_s \leftarrow q_{near}$;
while *true* **do**
  if $q_{target} = q_s$ **then**
    return q_s;
  else if $|q_{target} - q_s| > \left|q^{old}_s - q_{target}\right|$ **then**
    return q^{old}_s;
  end
  $q^{old}_s \leftarrow q_s$;
  $q_s \leftarrow q_s + \min(\Delta q_{step}, |q_{target} - q_s|) \frac{(q_{target} - q_s)}{|q_{target} - q_s|}$;
  $c \leftarrow$ GetConstraintValues(T, q^{old}_s);
  $\{q_s, c\} \leftarrow$ ConstrainConfig(q^{old}_s, q_s, c, NULL);
  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

It is important to note that TSR Chains inherently accommodate such mixing of real and virtual constraints. In fact a TSR Chain can consist of purely virtual or purely physical constraints. However, when planning with TSR Chains, special care must be taken to ensure that any physical joints (such as the door's hinge) be synchronized with their TSR Chain counterparts. This is done by including the configuration of any physical joints corresponding to elements of TSR Chains in the C-space searched by the planner (see Section V-C).

In the case that the physical constraints included in the TSR Chain form a redundant manipulator, we recommend a physical simulation of the movement of the end-effector from it's initial pose to $\mathbf{T}^0_{vee}$ as it is being pulled by the robot to find the resting configuration of the chain.

E. Notes on Implementation

Whenever we create a TSR Chain, we also create its virtual manipulator in simulation so that we can perform IK and get the location of the virtual end-effector. When we refer to the joint values of a TSR Chain, we are actually referring to the joint values of that TSR Chain's virtual manipulator. Also, to differentiate whether a TSR Chain should be used for sampling goals or constraining configurations or both, we specify how the chain should be used in its definition. When inputing TSR Chains into our planner, we specify which manipulator of the robot they correspond to as well as any physical DOF that correspond to elements of the chain.

V. The CBiRRT2 Algorithm

The CBiRRT2 algorithm is partly a combination of the CBiRRT algorithm [2] and the IKBiRRT algorithm [1]. CBiRRT2 (see Algorithm 1) takes into account constraints on the configuration of the robot during its path as well as constraints on the robot's goal configuration. The algorithm meets end-effector goal constraints by adding sampled roots to the backward tree. It meets end-effector pose constraints by projecting samples to the manifold corresponding to a given pose constraint. Constraints on the poses and goal locations of the end-effectors are specified as TSR Chains, which entails several important modifications from our previous work (Section V-B).

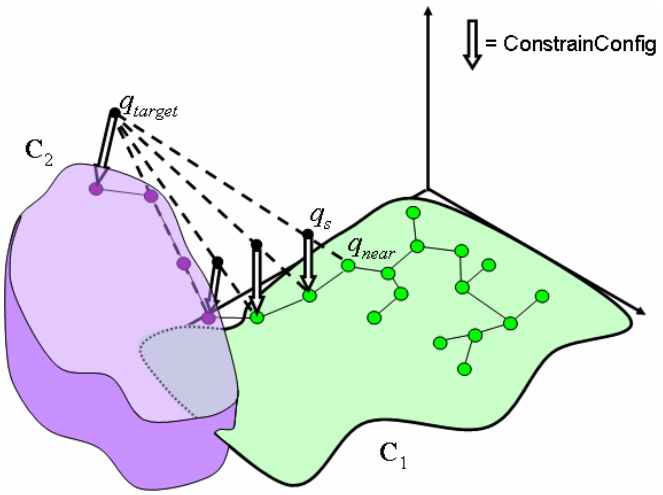


Fig. 4. Depiction of one **ConstrainedExtend** operation that moves across two constraint manifolds. The operation starts at q_{near}, which is a node of a search tree on constraint manifold $\mathbf{C}_1$ and iteratively moves toward q_{target}, which is a randomly-sampled configuration in C-space. Each step toward q_{target} is constrained using the ConstrainConfig function to lie on the closest constraint manifold.

A. Planner Operation

CBiRRT2 operates by growing two trees in the C-space of the robot. At each iteration, CBiRRT2 chooses between one

Algorithm 3: AddRoot(T)

```
for i = 1...m do
    ℂ ← GetTSRChainsForManipulator(i);
    {T^0_targ, c} ← SampleFromTSRChains(ℂ);
    Targets.AddTarget(T^0_targ, i);
end
{q_s, c} ← GetInitialGuess();
{q_s, c} ← ConstrainConfig(NULL, q_s, c, Targets);
if q_s ≠ NULL then
    T.AddVertex(q_s, c);
end
```

of two modes: exploration of the C-space using the two trees or sampling from a set of goal TSR Chains. The probability of choosing to sample is defined by the parameter P_{sample}.

If the algorithm chooses to sample, it calls the AddRoot function, which tries to inject a goal configuration into the backward tree T_{goal}. If the algorithm chooses to explore the C-space, one of the trees grows a branch toward a randomly-sampled configuration q_{rand} using the ConstrainedExtend function. The branch grows as far as possible toward q_{rand} but may be stalled due to constraint violation and will terminate at q^a_{reach}. The other tree then grows a branch toward q^a_{reach}, again growing as far as possible toward this configuration. If the other tree reaches q^a_{reach}, the trees have connected and a path has been found. If not, the trees are swapped and the above process is repeated.

The ConstrainedExtend function (see Algorithm 2) works by iteratively moving from a configuration q_{near} toward a configuration q_{target} with a step size of Δq_{step}. After each step toward q_{target}, the function checks if the new configuration q_s has reached q_{target} or if it is moving farther from q_{target}, in either case the function terminates. If the above conditions are not true then the algorithm takes a step toward q_{target} and passes the new q_s to the ConstrainConfig function, which is problem-specific. If ConstrainConfig is able to project q_s to a constraint manifold, the new q_s is added to the tree and the stepping process is repeated. Otherwise, ConstrainedExtend terminates (see Figure 4). ConstrainedExtend always returns the last configuration reached by the extension operation. The c vector is a vector of TSR Chain joint values of all TSR Chains. Every q_s has a corresponding c which is stored along with q_s in the tree. We store the c vector so that the ConstrainConfig function has a good initial guess of the TSR chain joint values when taking subsequent steps. This greatly decreases the time used by the inverse-kinematics solver inside ConstrainConfig.

The SmoothPath function uses the "short-cut" smoothing method to iteratively shorten the path using the ConstrainedExtend function. It is the same as that used in CBiRRT [2].

Note that CBiRRT2 can also be seeded with multiple start and goal configurations (Q_s/Q_g). If no goals are specified, the AddRoot function will insert the first goal into the backward tree. In fact, the AddRoot function can be called for both the start and the goal trees, if this is desired.

Algorithm 4: ConstrainConfig(q^{old}_s, q_s, c, Targets)

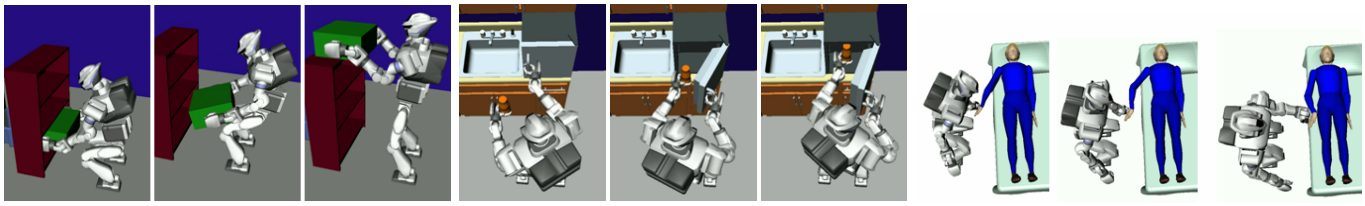
```
CheckDist = false;
if Targets = NULL then
    CheckDist = true;
    for i = 1...m do
        ℂ ← GetTSRChainsForManipulator(i);
        T^0_s ← GetEndEffectorTransform(q_s, i);
        {T^0_targ, c} ← GetClosestTransform(ℂ, T^0_s, c);
        Targets.AddTarget(T^0_targ, i);
    end
end
q_s ← UpdatePhysicalConstraintDOF(q_s, c);
q_s ← ProjectConfig(q_s, Targets);
if (q_s = NULL or
   (CheckDist and |q_s − q^old_s| > 2Δq_step)  then
    return NULL;
end
return {q_s, c};
```

B. Accounting for TSR Chains

Accounting for TSR chains is done in the AddRoot and ConstrainConfig functions. When CBiRRT2 chooses to sample a goal configuration, it calls the AddRoot function (see Algorithm 3). This function retrieves the relevant set of TSR Chains for each manipulator and samples a target transform for each manipulator using the SampleFromTSRChain function, which is an implementation of the methods described in Section IV-B. It then forms an initial guess of the robot's joint values and c and calls the ConstrainConfig function. In practice we usually use the initial configuration of the robot and vector of zeros for c as the guess but these can be randomized as well. If the ConstrainConfig does not return null, the resulting q_s and corresponding c are added to the tree.

The ConstrainConfig function is problem-specific, an example of a ConstrainConfig function that considers *only* TSR Chains is given in Algorithm 4. If ConstrainConfig is not passed a set of targets (i.e. it is called from ConstrainedExtend instead of AddRoot), then it generates a set of targets for each manipulator using the GetClosestTransform function, which is an implementation of the first two steps of the IK handshaking method described in Section IV-C. Note that this function also updates the c vector with the joint values of the TSR Chain that generated the closest transform. The c values for the TSR Chains that did not yield the closest transform to $\mathbf{T}^0_s$ are not updated. After the target transforms for each manipulator are obtained, ProjectConfig projects the configuration of the robot using standard inverse-kinematics algorithms based on the Jacobian pseudo-inverse to produce a q_s which meets the constraints posed by the TSR Chains. This completes the third and final step of the IK handshaking process.

If ConstrainConfig was called by AddRoot, the distance between q^{old}_s and q_s is irrelevant. However we do not wish for q_s to be too far from q^{old}_s when extending using Con-



(a) (b) (c)

Fig. 5. Snapshots from paths for the three examples. (a) Closed chain kinematics. (b) Simultaneous constraints and goal sampling. (c) Manipulating a passive chain.

strainedExtend because configurations between q_s^{old} and q_s are likely to violate the constraints. Thus we enforce a small step size to reduce deviation from the constraints between nodes.

In most situations, we are also interested in satisfying other constrains such as balance and collision. Checks for these constraints should be inserted at line 14 of ConstrainConfig.

C. Augmenting Configuration with States of Physical DOF

Because TSR chains can specify constraints corresponding to physical degrees of freedom of objects in the world (such as the hinge of a door), we have to account for physical DOF when checking collision and measuring distances in the C-space. To achieve this, we include the configuration of all physical DOF in the configuration vector q. We set these DOF by extracting their values from c using the UpdatePhysicalConstraintDOF function. This done on line 11 of the ConstrainConfig function. Note that these DOF are not affected by the subsequent ProjectConfig function.

VI. Results

We conducted several experiments to demonstrate the versatility of the TSR framework and to measure the performance of CBiRRT2 in a variety of situations. We used the HRP3 robot, which has two 6DOF legs, two 7DOF arms, and 2DOF in the waist, for a total of 28DOF. The kinematic tree of the robot was rooted at the right leg. We used the ConstrainConfig function of Algorithm 4 and include balance and collision constraints on line 14 so all paths produced by the planner are guaranteed to be collision-free and quasi-statically balanced (the projection of the center of mass is in the support polygon). All experiments were conducted on a 2.4GHz CPU with 4GB of RAM and runtimes for 30 runs of each problem are shown in Table 1. Please see a video of the experiments at:

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

A. Closed Chain Kinematics

Some researchers approach the problem of planning with closed-chain kinematics by implementing specialized projection operators [15] [16] or sampling algorithms [17]. However, in the TSR framework no special additions are required.

Consider the problem shown in Figure 5a. The task is for the robot to pick up the box from the bottom shelf of the book shelf and place it on top of the book shelf. There are two closed chains which must be enforced by the planner; the legs and arms. In this problem we get the goal-configuration of the robot from IK on the box target.

We define three TSR Chains, each containing one TSR. The first TSR is assigned to the left leg of the robot and allows no deviation from the current left-foot location (i.e. $\mathbf{B}^w = \mathbf{0}_{6\times 2}$). The second and third TSRs are assigned to the left and right arms and are defined relative to the location of the box (i.e. the 0 frame of $\mathbf{T}_w^0$ is the frame of the box). The bounds are defined such that the hands will always be holding the sides of the box at the same locations ($\mathbf{B}^w = \mathbf{0}_{6\times 2}$). The geometry of the box is "attached" to the right hand.

The result of this construction is the following: When ConstrainedExtend generates a new q_s, the box moves with the right hand and the frame of the box changes thus breaking the closed-chain constraint. This q_s is passed to ConstrainConfig, which projects q_s to meet the constraint (i.e. moving the left arm). The same process happens simultaneously for the left leg of the robot as well.

We implemented this example in simulation and on the real HRP3 robot. On the real robot, the task was to stack two boxes in succession, snapshots from the execution of the plan can be seen in Figure 6. The experiments on the robot show that we can enforce stringent closed-chain constraints using the CBiRRT2 planner and the TSR framework.

B. Simultaneous Constraints and Goal Sampling

The task in this problem is to place a bottle held by the robot into a refrigerator (see Figure 5b). The use of TSR Chains is important here, because it allows the right arm of the robot to rotate about the handle of the refrigerator, which gives the robot more freedom when opening the door. We assume that the grasp cages the door handle (as in [18]) so the end-effector can rotate about the handle without the door escaping. P_{sample} was set to 0.1 for this problem.

There are four TSR Chains defined for this problem. The first is the TSR Chain(1 element) for the left leg, which is the same as in Example A. This TSR Chain is marked for both sampling goal configurations and constraining the configuration of the robot. The second TSR Chain(2 element) is defined for the right arm and is described in Section IV-A. This chain is also marked for both sampling goal configurations and constraining the configuration of the robot. The third TSR Chain(1 element) is defined for the left arm and constrains the robot to disallow titling of the bottle during the robot's motion. This chain is used only for constraining the robot's configuration. The final TSR Chain(1 element) is also defined for the left arm and represents allowable placements

Fig. 6. Snapshots from the execution of the box stacking task using the HRP3 robot.

of the bottle inside the refrigerator. Its $\mathbf{B}^w$ has freedom in x and y equal to the refrigerator width and length, and no other freedoms. This chain is only used for sampling goals.

The result of this construction is that the robot simultaneously samples a target bottle location and wrist position for its right arm when sampling goal configurations. It is important to note is that we can be rather sloppy when defining TSRs for goal sampling. Observe that many samples from the right arm's TSR chain will leave the door closed or marginally open, thus placing the left arm into collision if it is reaching inside the refrigerator. However, this is not an issue for the planner because it can always sample more goal configurations and the collision constraint is included in ConstrainConfig.

C. Manipulating a Passive Chain

The task in this problem is for the robot to assist in placing a disabled person into bed (see Figure 5c). The robot is to move the person's right hand to a specified point near his body. The person's arm is assumed to be completely passive and the kinematics of the arm (as well as joint limits) are assumed to be known. In this problem, we get the goal-configuration of the robot from IK on the target pose of the person's hand. The robot's grasp of the person's hand is assumed to be rigid.

There are two TSR Chains defined for this problem, both of which are used to constrain the robot's configuration. The first is the TSR Chain(1 element) for the left leg, which is the same as in Example A. The second is a TSR Chain(6 element) defined for the person's arm. Every element of this chain corresponds to a physical DOF of the person's arm. Note that since the arm is not redundant, we do not need to perform any special IK to ensure that the configuration of the person matches what it would be in the real world.

The result is that the person's arm will track the robot's left hand. Since the configuration of the person's arm is included in q, there cannot be large discontinuities in the person's arm configuration (i.e. elbow-up to elbow-down) because such configurations are distant in C-space.

	Mean	Std. Dev
Closed Chain Kinematics	4.21s	2.00s
Simultaneous Constraints and Goal Sampling	1.54s	0.841s
Manipulating a Passive Chain	1.03s	0.696s

TABLE I: RUNTIMES FOR EXAMPLE PROBLEMS

VII. CONCLUSION

We have presented an efficient approach to generating paths for humanoids and other robotic manipulators that uses TSR Chains to specify manipulation tasks. A given task can entail any number of TSR Chains, which can consists of any number of TSRs. TSR Chains can be sampled and the distance to them can be determined efficiently, which allows them to represent both constraints on and goal poses of a robot's end-effector. The CBiRRT2 planner uses TSR Chains to plan for a variety of pose-constrained manipulation tasks. It takes only several seconds to plan for complex manipulation tasks, demonstrating the efficiency of the CBiRRT2 and the TSR framework.

VIII. ACKNOWLEDGEMENTS

Dmitry Berenson was partially supported by Intel Labs Pittsburgh and by NSF Grant No. EEC-0540865.

REFERENCES

[1] D. Berenson, S. S. Srinivasa, D. Ferguson, A. Collet, and J. Kuffner, "Manipulation planning with workspace goal regions," in *ICRA*, 2009.
[2] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. Kuffner, "Manipulation planning on constraint manifolds," in *ICRA*, 2009.
[3] M. Stilman, J. U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *ICRA*, 2007, pp. 3327–3332.
[4] 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, pp. 2041–2046.
[5] E. Drumwright and V. Ng-Thow-Hing, "Toward interactive reaching in static environments for humanoid robots," in *IROS*, 2006, pp. 846–851.
[6] M. Vande Weghe, D. Ferguson, and S. S. Srinivasa, "Randomized path planning for redundant manipulators without inverse kinematics," in *Humanoids*, 2007, pp. 477–482.
[7] D. Bertram, J. Kuffner, R. Dillmann, and T. Asfour, "An integrated approach to inverse kinematics and path planning for redundant manipulators," in *ICRA*, 2006, pp. 1874–1879.
[8] S. LaValle and J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *WAFR*, 2000.
[9] L. Sciavicco and B. Siciliano, *Modeling and Control of Robot Manipulators*, 2nd ed. Springer, 2000, pp. 96–100.
[10] 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.
[11] Y. Koga, K. Kondo, J. Kuffner, and J. claude Latombe, "Planning motions with intentions," in *SIGGRAPH*, 1994.
[12] K. Yamane, J. Kuffner, and J. Hodgins, "Synthesizing animations of human manipulation tasks," in *SIGGRAPH*, 2004.
[13] Z. Yao and K. Gupta, "Path planning with general end-effector constraints: using task space to guide configuration space search," in *IROS*, 2005, pp. 1875–1880.
[14] M. Stilman, "Task constrained motion planning in robot joint space," in *IROS*, 2007, pp. 3074–3081.
[15] J. H. Yakey, S. M. LaValle, and L. E. Kavraki, "Randomized path planning for linkages with closed kinematic chains," *ICRA*, vol. 17, no. 6, pp. 951–958, 2001.
[16] K. Hauser, "Motion planning for legged and humanoid robots," Ph.D. dissertation, Stanford University, September 2008.
[17] J. Cortes and T. Simeon, "Sampling-based motion planning under kinematic loop-closure constraints," in *WAFR*, 2004.
[18] R. Diankov, S. S. Srinivasa, D. Ferguson, and J. Kuffner, "Manipulation planning with caging grasps," in *Humanoids*, 2008, pp. 285–292.