IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2012

Manipulation Planning with Soft Task Constraints

Tobias Kunz and Mike Stilman

***Abstract*— We present a randomized configuration space planner that enforces soft workspace task constraints. A soft task constraint allows an interval of feasible values while favoring a given exact value. Previous work only allows for enforcing an exact value or an interval without a specific preference. Soft task constraints are a useful concept in everyday life. For example when carrying a container of liquid we want to keep it as close to the upright position as possible but want to be able to tilt it slightly in order to avoid obstacles. This paper introduces the necessary algorithms for handling such constraints, including projection methods and useful representations of everyday constraints. Our algorithms are evaluated on a series of simulated benchmark problems and shown to yield significant improvement in constraint satisfaction.**

I. Introduction

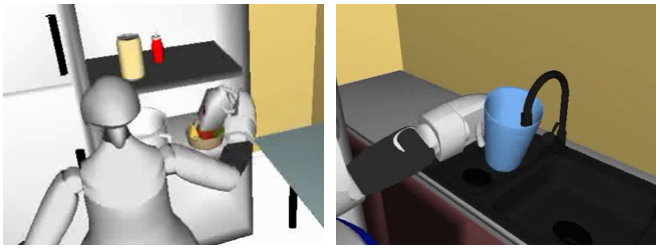
In this paper we introduce a randomized configuration space planner that enforces soft task constraints. Hard constraints divide configurations into feasible and infeasible ones. In contrast, soft constraints also specify a preference within the feasible region such as staying away from the boundary. In everyday life, a container of liquid should be kept as upright as possible to minimize the risk of spills. Yet, there might be situations where we need to tilt the container slightly to avoid an obstacle. A slight tilt is less likely to lead to a spill.

Examples of such problems, shown in Fig. 1, include taking a milk container out of a fridge and filling a pitcher with water under a faucet. When removing the pitcher from underneath the faucet, the faucet might interfere and force us to tilt the pitcher. The more we tilt, the less water we can add without spilling. Even robot balance within a support region is safer if the robot's center of mass stays close to the center of the region. We present a bidirectional RRT planner [1] with a path shortener for problems with soft task constraints. Although we use an RRT planner to demonstrate the effectiveness of our strategy, it can also be applied to other sampling-based planners such as PRM [2].

A. Related Work

First-order retraction was introduced in [3, 4] as a method to project a configuration space sample onto a constraint manifold induced by workspace task constraints. It was shown that first-order retraction is more efficient than randomized gradient descent (RGD) [5]. Task constraints are defined by declaring the coordinates of a task frame as either constrained or unconstrained. In [4] a translation or rotation around a coordinate axis can only be fixed completely within ϵ deviation. For the previous example this means we can only constrain the container of liquid to stay upright. If there is no path satisfying this constraint, planning fails even if there was a solution that requires a small tilt.

The authors are with the Center for Robotics and Intelligent Machines at the Georgia Institute of Technology, Atlanta, GA 30332, USA. tobias@gatech.edu, mstilman@cc.gatech.edu



(a) Taking milk out of the fridge (b) Filling a pitcher with water

Fig. 1: Application examples of soft task constraints

In [6], constraints are presented that allow for an interval of values for translations and rotations. This means we could allow the container to be tilted up to a maximum angle. [6] also presents a task-constrained path shortening method. However, intervals are still hard constraints. All configurations satisfying the constraint are equally good with no bias toward the center of the constraint. In fact there is a bias toward the boundary of the constraint, since the infeasible samples are projected toward the nearest constraint boundary. This leads to the undesirable case where the robot may keep a container close to the boundary limit.

The task-constrained planning methods introduced in [3, 4, 6] have been applied to various manipulation scenarios, e. g. [7, 8]. In contrast, [9] samples and searches for nearest-neighbors directly in task space in order to better guide the RRT growth in very-high-dimensional configuration spaces. The RRT is grown in configuration space toward the sample using a projection method similar to the one introduced in [3, 4]. [10] generalizes the constraint representation beyond constraints on the end-effector motion along axes of the task frame and applies it to humanoid whole-body motion planning. However, all these methods focus on hard constraints.

[11] improves the efficiency of constrained sampling by sampling in the tangent-space of the constraint manifold instead of the full configuration space. Samples are projected back on the manifold if they are too far away. [12] does not use projection at all but instead builds an atlas consisting of charts that locally approximate the constraint manifold. The atlas as an approximation of the constraint manifold is then searched for a path. However, both [11] and [12] are only applicable to exact constraints that are represented by a constraint manifold. Neither of the two represents the notion of getting as close as possible to the manifold while accepting all configurations within a certain distance.

B. Contribution and Overview

We propose a projection method that projects a configuration sample as close to the desired value of translation or rotation

as possible. Even when the optimal value cannot be reached, we accept the projected sample if it is within the given constraint limits. The robot holds the container of liquid upright whenever it can and allows tilt when necessary to find a solution. No existing algorithms achieve this behavior.

We also introduce a new representation for the rotation in task coordinates, which is better suited to define a limit on the angle between an object and a fixed axis, such as the vertical. Finally, we present a conversion from task coordinate error to task frame error that is simpler and numerically more stable than the one proposed in [4].

Section II describes how we represent soft constraints. Section III introduces our algorithm for creating soft-constrained samples. Section IV describes the high-level planning algorithm we use to demonstrate the effectiveness of our soft-constrained sampling algorithm. In Section V we present experimental results from running the planning algorithm in two different scenarios.

II. Constraint Representation

The homegeneous transform of frame b relative to a is

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

The task constraint is given by defining limits on allowable translations and rotations relative to the task frame. The homogeneous transform T_t^0 specifies the pose of the task frame relative to the world frame. The task frame can be stationary in the world, e. g. at a door hinge, or it can change with the pose of the object. The pose of the object relative to the task frame is given by

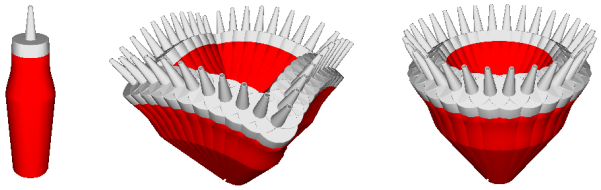
$$\boldsymbol{T}_{obj}^t = (\boldsymbol{T}_t^0)^{-1}\boldsymbol{T}_{obj}^0 \tag{2}$$

The task constraint limits the displacement of the object from the task frame. We use task coordinates to represent the displacement of the object from the task frame. The choice of these coordinates is task-specific. As we can only limit task coordinates individually, the task coordinates have to be chosen in such a way that they line up with what we want to limit. Previous papers [4] and [6] use Cartesian coordinates for translation and X-Y-Z Euler angles for rotation. We also use Cartesian coordinates for translation but use a Z-Y-Z Euler angle convention, which is better suited to represent an angular displacement from one fixed axis.

Consider the case of constraining the tilt angle of a bottle as shown in Figure 2a. The z-axis is pointing up. When using the X-Y-Z Euler angles, we need to constrain the rotations around both the x- and y-axes. But constraining those 2 rotations does not lead to a consistent tilt angle limit. Figure 2b shows the constraint boundary for a limit of $\pm 30°$ for both rotations. In contrast, when using Z-Y-Z Euler angles, the angle of rotation around the y-axis equals the tilt angle and can be easily limited as shown in Figure 2c.

Using Z-Y-Z Euler angles, ϕ, θ and ψ define the rotation of the object relative to the task frame as follows.

$$\boldsymbol{R}_{obj}^t = R_z(\phi)R_y(\theta)R_z(\psi) \tag{3}$$



(a) bottle (b) X-Y-Z (c) Z-Y-Z

Fig. 2: Constraint bounds for different Euler representations

where $R_x(\phi)$ represents the rotation matrix for a rotation around the x-axis by an angle of ϕ. Given a rotation matrix we can calculate the task coordinates representing the rotation as follows. Since we use a different Euler angle convention, these equations differ from [4] and [6].

$$\psi = \operatorname{atan2}(\boldsymbol{R}_{3,2}, -\boldsymbol{R}_{3,1}) \tag{4}$$

$$\theta = \arccos(\min(1, \boldsymbol{R}_{3,3})) \tag{5}$$

$$\phi = \operatorname{atan2}(\boldsymbol{R}_{2,3}, \boldsymbol{R}_{1,3}) \tag{6}$$

with $\boldsymbol{R} = \boldsymbol{R}_{obj}^t$ and $\boldsymbol{R}_{i,j}$ representing the value at row i and column j of $\boldsymbol{R}$. The minimum is taken here in order to accommodate for numerical inaccuracies. The full task deviation vector is given as

$$\boldsymbol{\Delta x} = \begin{bmatrix} \boldsymbol{t}_{obj}^t \\ \psi \\ \theta \\ \phi \end{bmatrix} \tag{7}$$

Given any task coordinate representation consisting of n coordinates (in our case $n = 6$), we can select the constrained coordinates. We do so by defining a diagonal $n \times n$ constraint selection matrix $\boldsymbol{C}$. The i^{th} diagonal element of $\boldsymbol{C}$ is 1 if the i^{th} task coordinate is constrained, otherwise it is 0. We also define limits on the constrained task coordinates

$$\boldsymbol{\Delta x}_{\min} \leq \boldsymbol{C}\boldsymbol{\Delta x} \leq \boldsymbol{\Delta x}_{\max} \tag{8}$$

$$\boldsymbol{\Delta x}_{\min} \leq \boldsymbol{0} \leq \boldsymbol{\Delta x}_{\max} \tag{9}$$

Without loss of generality, we assume that the favored value for task coordinates is 0. In general, we could move the task frame such that it is at 0. We term this value the center of the constraint interval, even when it is not literally centered.

Note that we use both a constraint selection matrix $\boldsymbol{C}$ and limit vectors $\boldsymbol{\Delta x}_{\min}$ and $\boldsymbol{\Delta x}_{\max}$, while [4] only uses a constraint selection matrix and [6] only uses limit vectors. [4] does not need limit vectors because it only allows for exact constraints. [6] does not need a selection matrix because coordinates can be unconstrained by setting the limits to $-\infty$ and ∞. That does not work in our case, because even though we might want to allow all possible values for a coordinate we might still favor a specific value.

III. Constrained Sampling

We constrain a configuration by using a gradient-descent method that repeatedly moves the configuration closer to the constraint. The gradient is given as joint space error, calculated as a local linear approaximation using the Jacobian pseudo-inverse. Section III-A describes how to calculate the

joint space error. Section III-B describes how this information is used to project a sample as close to the center of the constraint interval as possible.

A. Calculating Joint Space Error

For a given robot configuration we calculate the pose of the object, which is rigidly attached to the robot, using forward kinematics. The pose of the object relative to the task frame is then given by

$$\boldsymbol{T}^t_{obj}(\boldsymbol{q}) = (\boldsymbol{T}^0_t)^{-1}\boldsymbol{T}^0_{obj}(\boldsymbol{q}) \tag{10}$$

Using Eq. 4 - 7, we can get the task coordinate representation $\boldsymbol{\Delta x}$ from $\boldsymbol{T}^t_{obj}$. The task error $\boldsymbol{\Delta x_{err}}$ is easily calculated by applying the constraint selection matrix $\boldsymbol{C}$.

$$\boldsymbol{\Delta x}_{err}(\boldsymbol{q}) = \boldsymbol{C}\boldsymbol{\Delta x}(\boldsymbol{q}) \tag{11}$$

Given the task error $\boldsymbol{\Delta x}_{err}$, we want to move the robot such that the error is reduced. The Jacobian $\boldsymbol{J}(\boldsymbol{q})$ gives a mapping from joint space velocities to work space velocities in the world frame. Like [4] and [6] we use the right pseudo-inverse

$$\boldsymbol{J}^\dagger = \boldsymbol{J}^T\left(\boldsymbol{J}\boldsymbol{J}^T\right)^{-1} \tag{12}$$

of the Jacobian to map work space velocities into joint space velocities. Since the task error is given relative to the task frame we need to transform it into the world frame before multiplying it with the Jacobian pseudo-inverse.

$$\boldsymbol{J}^\dagger_t(\boldsymbol{q}) = \boldsymbol{J}^\dagger(\boldsymbol{q})\begin{bmatrix}\boldsymbol{R}^0_t & 0\\ 0 & \boldsymbol{R}^0_t\end{bmatrix} \tag{13}$$

The task-frame Jacobian pseudo-inverse still does not map the task error into a joint space velocity that locally reduces the task error in task coordinates. Because Euler angles are applied sequentially, a rotation around one axis affects the location of the axes of rotations applied before. Thus, in order to reduce the task error locally, we need to rotate around the current position of the rotation axes corresponding to the Euler angles. In order to find these axes we multiply each original Euler axis with rotation matrices representing the rotations applied afterwards. Thus, the vector of instantaneous rotations around the task frame axes ω is found by multiplying each instantaneous Euler angle by the corresponding original axis relative to the task frame as well as with the rotation matrices of the Euler rotations applied afterwards.

$$\omega = R_z(\phi)R_y(\theta)\hat{\boldsymbol{z}}\psi + R_z(\phi)\hat{\boldsymbol{y}}\theta + \hat{\boldsymbol{z}}\phi \tag{14}$$

By collecting the factors in front of the Euler angles and by adding the linear part, we get the transformation matrix

$$\boldsymbol{E}^{-1} = \begin{bmatrix}\boldsymbol{I}_3 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0}\\ \boldsymbol{0} & \hat{\boldsymbol{z}} & R_z(\phi)\hat{\boldsymbol{y}} & R_z(\phi)R_y(\theta)\hat{\boldsymbol{z}}\end{bmatrix} \tag{15}$$

$$= \begin{bmatrix}\boldsymbol{I}_3 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0}\\ \boldsymbol{0} & \sin\theta\cos\phi & -\sin\phi & 0\\ \boldsymbol{0} & \sin\theta\sin\phi & \cos\phi & 0\\ \boldsymbol{0} & \cos\theta & 0 & 1\end{bmatrix} \tag{16}$$

Algorithm 1: NewConfig($q_{\text{target}}, q_{\text{near}}, \delta$)

1 $q_{\text{step}} \leftarrow q_{\text{near}} + \min\{\delta, ||q_{\text{target}} - q_{\text{near}}||\}\frac{q_{\text{target}} - q_{\text{near}}}{||q_{\text{target}} - q_{\text{near}}||}$;
2 $q_{\text{new}} \leftarrow NULL$; $\Delta x^{prev}_{err} \leftarrow \infty$;
3 **while** $q_{min} \leq q_{step} \leq q_{max}$ **and** *CollisionFree*(q_{near}, q_{step}) **do**
4 $\quad \Delta x \leftarrow \Delta x(q_{\text{step}})$;
5 $\quad \Delta x_{err} \leftarrow C\Delta x$;
6 $\quad$ **if** $\Delta x_{err} \geq \Delta x^{prev}_{err}$ **then break**;
7 $\quad \Delta x^{prev}_{err} \leftarrow \Delta x_{err}$;
8 $\quad q_{\text{new}} \leftarrow q_{\text{step}}$;
9 $\quad$ **if** $\Delta x_{err} \leq \epsilon$ **then break**;
10 $\quad \Delta q_{\text{err}} \leftarrow J^\dagger(q_{\text{step}})\begin{bmatrix}R^0_t & 0\\ 0 & R^0_t\end{bmatrix} E^{-1}(\Delta x)\, \Delta x_{err}$;
11 $\quad q_{\text{step}} \leftarrow q_{\text{step}} - \min\{\delta, ||\Delta q_{\text{err}}||\}\frac{\Delta q_{\text{err}}}{||\Delta q_{\text{err}}||}$;
12 **end**
13 **if** $\Delta x_{min} - \epsilon \leq \Delta x_{err} \leq \Delta x_{max} + \epsilon$ **and** $\left(||q_{target} - q_{new}|| + \epsilon < ||q_{target} - q_{near}||\right.$ **or** $\left.||\Delta x_{err}|| < ||C\Delta x(q_{near})||\right)$ **then**
14 $\quad$ **return** q_{new};
15 **else**
16 $\quad$ **return** NULL;
17 **end**

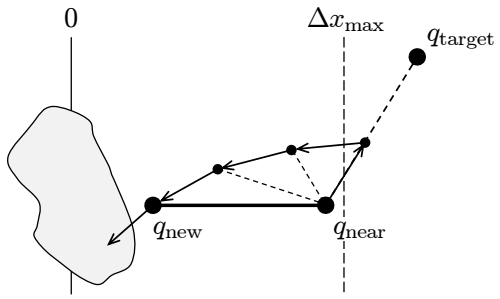


Fig. 3: Visualization of Algorithm 1

The Jacobian relative to the task coordinates of the current pose of the object is given by

$$\boldsymbol{J}^\dagger_{\boldsymbol{\Delta x}}(\boldsymbol{q}) = \boldsymbol{J}^\dagger_t(\boldsymbol{q})\boldsymbol{E}^{-1}(\boldsymbol{\Delta x}(\boldsymbol{q})) \tag{17}$$

The transformation matrix is called $\boldsymbol{E}^{-1}$ rather than $\boldsymbol{E}$ in order to be consistent with the naming in [4]. In [4] the task coordinate Jacobian pseudo-inverse is defined as $\boldsymbol{J}^\dagger_{\boldsymbol{\Delta x}}(\boldsymbol{q}) = (\boldsymbol{E}(\boldsymbol{q})\boldsymbol{J}_t(\boldsymbol{q}))^\dagger$. This is mathematically the same as our definition but has two drawbacks. First, the derivation is more complicated because $\boldsymbol{E}$ is defined as $(\boldsymbol{E}^{-1})^{-1}$. Second, $(\boldsymbol{E}^{-1})^{-1}$ as shown in the appendix of [4] is not well defined as it contains fractions with a sine or cosine as denominator. The denominator might become zero. In our case this happens at the favored upright object pose.

Combining Eq. 13 and 17, the joint error $\boldsymbol{\Delta q}_{err}$ based on a local approximation is given as

$$\boldsymbol{\Delta q}_{\text{err}}(\boldsymbol{q}) = \boldsymbol{J}^\dagger(\boldsymbol{q})\begin{bmatrix}\boldsymbol{R}^0_t & 0\\ 0 & \boldsymbol{R}^0_t\end{bmatrix}\boldsymbol{E}^{-1}(\boldsymbol{\Delta x}(\boldsymbol{q}))\, \boldsymbol{\Delta x}_{\text{err}}(\boldsymbol{q}) \tag{18}$$

Eq. 18 is found in line 10 of Algorithm 1.

Algorithm 2: BiRRT($q_1, q_2, \delta, n_{\max}$)

```
1  T_1.AddNode(q_1, NULL); T_2.AddNode(q_2, NULL);
2  a ← 1; b ← 2;
3  while Size(T_1) + Size(T_2) < n_max do
4      q_rand ← RandomConfig();
5      q^a_near ← NearestNeighbor(T_a, q_rand);
6      if q^a_new ← NewConfig(q_rand, q^a_near) then
7          T_a.AddNode(q^a_new, q^a_near);
8          q^b_near ← NearestNeighbor(T_b, q^a_new);
9          while q^b_new ← NewConfig(q^a_new, q^b_near, δ) do
10             if q^a_new = q^b_new then
11                 return ExtractPath(T_1, T_2);
12             end
13             T_b.AddNode(q^b_new, q^b_near);
14             q^b_near ← q^b_new;
15         end
16     end
17     Swap(a, b);
18 end
19 return NULL;
```

B. Projection Toward the Constraint

Instantaneous joint-space error is used to move the robot closer to the task constraint. [4] [6] subtract the whole joint space error from the current configuration. [4] moves all the way to the center of the constraint while [6] moves to the boundary of the constraint. We move to the center of the constraint but in small steps. Even inside the constraint boundaries we continue toward the center until we hit an obstacle. This is what makes our constraint enforcement soft.

In line 11 of Algorithm 1 the configuration is moved toward the constraint by a small step δ. We repeat this until either we hit an obstacle, reach a joint limit or reach the center of the constraint. We also abort if a step does not get us closer to the task constraint (line 6). This might happen because the step is calculated based on a linear approximation. Without this check the algorithm might not terminate. In line 13 we check whether the last collision-free configuration satisfies the task constraint. We also check whether we made progress toward the target configuration or the constraint. This is necessary to make sure the higher-level planning algorithm terminates. If so, we return the last configuration, which is as close to the center of the constraint as we could get. Algorithm 1 is visualized in Figure 3.

This algorithm results in numerous collision checks, since we check for collision at every step toward the constraint. The efficiency of the algorithm could be improved by deferring the collision checks. We could step toward the constraint without collision checks. Once the center of the constraint is reached, we could search along the projection path for the last collision-free configuration using bisection. This has negligible effect on the resulting path of the planner and thus not evaluated in our benchmarks.

IV. Planning Algorithm

Our planning algorithm consists of two parts. A bidirectional RRT planner and a path shortener. The RRT planner shown in Algorithm 2 is identical to [1]. We only replace the NewConfig function with the one in Algorithm 1.

Algorithm 3: LocalPlanner(q_1, q_2, δ)

```
1  L_1 ← [q_1]; L_2 ← [q_2];
2  a ← 1; b ← 2;
3  while ||Back(L_1) − Back(L_2)|| > δ do
4      if newConfig(q_new, Back(L_b), Back(L_a)) then
5          L_a ← Concat(L_a, [q_new]);
6          progress ← true;
7      else if progress then
8          progress ← false;
9      else
10         return NULL;
11     end
12     Swap(a, b);
13 end
14 return Concat(L_1, Reverse(L_2));
```

Algorithm 4: ShortenPath(P, δ, n)

```
1  for 1 to n· Size(P) do
2      repeat
3          i ← RandomInt(1, Size(P));
4          j ← RandomInt(1, Size(P));
5      until i < j − 1;
6      if S ← LocalPlanner(P_i, P_j, δ) then
7          P ← Concat([P_1,...,P_{i−1}], S,
           [P_{j+1},...,P_{Size(P)}]);
8      end
9  end
```

The path shortener shown in Algorithm 4 repeatedly selects two nodes on the path randomly and tries to shortcut between them. The shortcutting is shown in Algorithm 3. Unlike in [6] the shortcutting is bidirectional. We are trying to extend both nodes toward each other. This is necessary because otherwise we could never shortcut between two nodes that are not at the center of the constraint. We would extend toward the center of the constraint and then in the direction of the other node, but we would never get close to the other node because it is not in the center of the constraint.

V. Experimental Results

We evaluate our algorithm by applying it to two different scenarios. In Scenario ,1 shown in Figure 4, the task is to take the red bottle out of the shelf an put it on the table. The bottle is blocked by other objects such that it cannot be removed from the shelf in an upright position. We repeat this pick-and-place operation 100 times. In Scenario 2, shown in Figure 6, there are no obstacles and the bottle can be moved in an upright pose. The robot executes 10 randomly sampled pick-and-place operations. Locations that do not yield valid grasps due to IK or geometry are rejected. The 20 pick and place locations are shown in Figure 6. We repeat every pick-and-place operation 10 times resulting in a total of 100 trials.

The motivation for this paper is to design an algorithm that tilts the bottle as necessary in Scenario 1 but keeps it

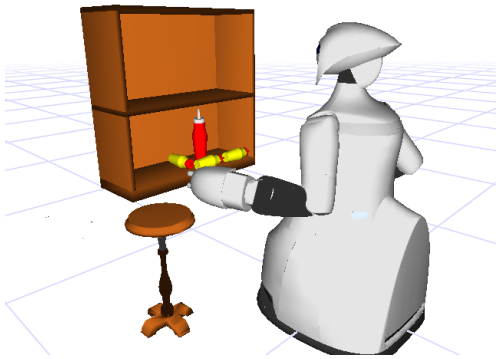


Fig. 4: Scenario 1

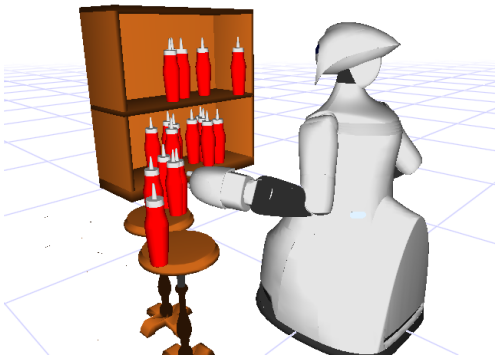


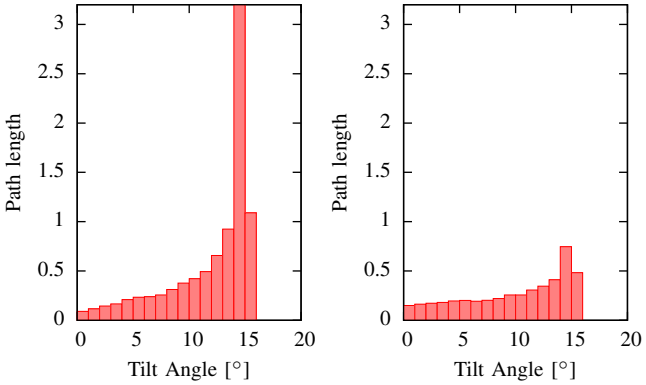
Fig. 6: Scenario 2

Tilt Limit	Constraint RRT — Short.	Max Tilt Angle	Average Tilt Angle	Path Length	Comp. Time
∞	None	180.0°	59.6°	4.1	1 s
	— None	179.9°	58.7°	2.8	2 s
0°	Hard	–	–	–	∞
	— Hard	–	–	–	–
15°	Hard	15.1°	12.3°	9.9	51 s
	— Hard	15.2°	10.0°	4.5	8 s
	— Soft	15.1°	4.3°	5.9	44 s
15°	Soft	15.1°	2.9°	12.8	281 s
	— Soft	15.1°	3.6°	5.6	58 s

TABLE I: Results for scenario 1

Tilt Limit	Constraint RRT — Short.	Max Tilt Angle	Average Tilt Angle	Path Length	Comp. Time
∞	None	178.3°	42.3°	4.1	0.6 s
	— None	145.7°	34.4°	2.7	1.7 s
0°	Hard	0.2°	0.0°	4.4	1.1 s
	— Hard	0.2°	0.0°	2.6	2.4 s
15°	Hard	15.1°	10.4°	3.7	1.2 s
	— Hard	15.1°	6.9°	2.3	1.9 s
	— Soft	15.1°	1.6°	2.7	8.0 s
15°	Soft	15.0°	1.8°	4.8	1.7 s
	— Soft	14.9°	1.6°	3.0	12.0 s

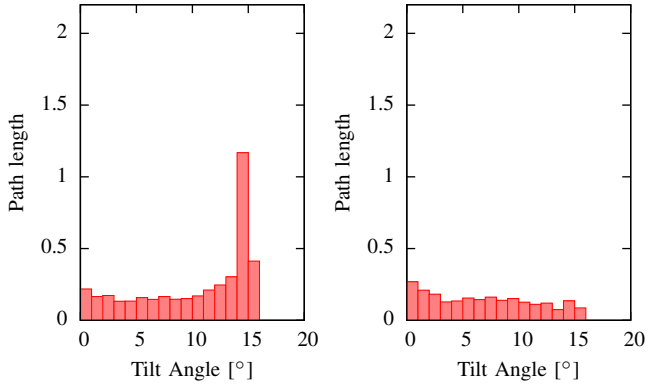
TABLE II: Results for scenario 2



(a) hard-constrained, unshortened (b) hard-constrained, shortened

(c) soft-constrained, unshortened (d) soft-constrained, shortened

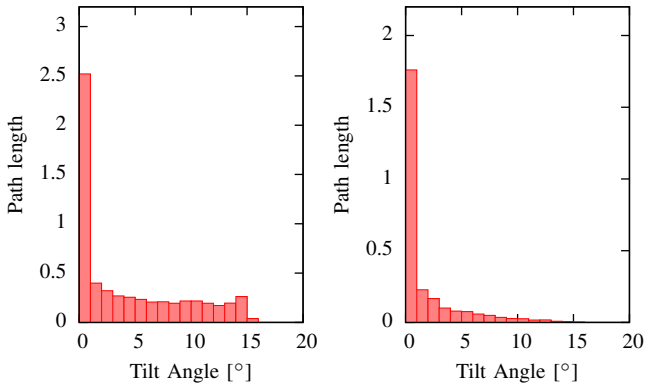
Fig. 5: Tilt angle distribution along the path for scenario 1



(a) hard-constrained, unshortened (b) hard-constrained, shortened

(c) soft-constrained, unshortened (d) soft-constrained, shortened

Fig. 7: Tilt angle distribution along the path for scenario 2



(a) Scenario 1 (b) Scenario 2

Fig. 8: Tilt angle distribution along the path using a hard-constrained RRT combined with soft-constrained shortening

upright in Scenario 2.

Table I and Table II show the results for the two scenarios. The tables show results for different constraint types: Unconstrained, exact-constrained, hard-constrained and soft-constrained. The first line of every section represents the path produced by the RRT planner without shortening. Subsequent lines in the same section represent paths after shortening has been applied to the RRT planner result. All numbers are averages over 100 planning runs. The maximum tilt angle is computed over all 100 runs. All runs were performed on an Intel Core i5-560M 2.66 GHz processor.

The unconstrained planner can solve both scenarios but leads to uncontrolled tilt angles. This can result in spilling liquid. The planner with an exact constraint on keeping the bottle upright, similar to the method presented in [4], successfully finds an all-upright path for Scenario 2 but fails in Scenario 1. For the hard- and soft-constrained planners we allow a tilt angle of up to $15°$. This is slightly more than what is necessary to remove the bottle from the shelf in Scenario 1. The hard-constrained planner successfully solves both scenarios and stays within the tilt angle limit of $15°$. Yet, it tilts the bottle significantly in Scenario 2, often keeping the bottle at the tilt limit. The soft-constrained planner also solves both scenarios but reduces the tilt angle drastically, especially in Scenario 2. However, it takes significantly longer to plan a path. This is because we perform numerous collision checks while moving the sampled configuration towards the constraint. The algorithm could be sped up by avoiding collision checks as described in Section III-B.

Figure 5 and Figure 7 show the tilt angle distribution of the paths for both scenarios, hard- and soft-constrained, unshortened and shortened. The graphs show that the hard-constrained planner [6] leads to a bias toward the tilt angle limit, whereas the soft-constrained planner effectively reduces the tilt angle. The soft-constrained planner does not always keep the bottle exactly upright in scenario 2. All configurations on the path produced by the soft-constrained planner that are not exactly upright are close to an obstacle or joint limit. Projecting them closer to the upright position would lead to a collision.

We also combined a hard-constrained RRT planner with soft-constrained shortening. We did that to be able to compare the hard- and soft-constrained path shorteners on their own starting from the same unshortened path. Also, this combination seems to be a good trade-off between path quality and computation time. Figure 8 shows the resulting tilt angle distribution of this combination for both scenarios.

VI. Conclusion

We presented a path planner that is able to enforce soft task constraints. Our planner improves over existing work. Unlike [4] it is able to consider a whole interval of values for a constrained quantity. Unlike [6] it is able to favor one value within the interval. We demonstrated the usefulness and effectiveness of our algorithm by showing that it can effectively keep a container of liquid near the upright position most of the time while still allowing some tilt in situations where tilting the container is unavoidable.

Acknowledgements

This work is supported by Toyota Motor Engineering & Manufacturing North America (TEMA). We thank Douglas Moore and Yasuhiro Ota (TEMA) for their insights on soft-constrained tasks and Michael Grey (GT) for implementing the examples shown in Fig. 1 and the corresponding parts of the accompanying video.

References

[1] J. Kuffner, J.J. and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2000.

[2] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, 1996.

[3] M. Stilman, "Task constrained motion planning in robot joint space," in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.

[4] M. Stilman, "Global manipulation planning in robot joint space with task constraints," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 576 –584, 2010.

[5] J. Yakey, S. LaValle, and L. Kavraki, "Randomized path planning for linkages with closed kinematic chains," *IEEE Transactions on Robotics and Automation*, 2001.

[6] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2009.

[7] S. Srinivasa, D. Ferguson, C. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. Weghe, "Herb: a home exploring robotic butler," *Autonomous Robots*, vol. 28, pp. 5–20.

[8] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2007.

[9] A. Shkolnik and R. Tedrake, "Path planning in 1000+ dimensions using a task-space voronoi bias," in *2009. Proc. of IEEE Int. Conf. on Robotics and Automation*, 2009.

[10] S. Dalibard, A. Nakhaei, F. Lamiraux, and J.-P. Laumond, "Whole-body task planning for a humanoid robot: a way to integrate collision avoidance," in *Proc. of IEEE-RAS International Conference on Humanoid Robots*, 2009.

[11] T. T. Um, B. Kim, C. Suh, and F. C. Park, "Tangent space rrt with lazy projection: An efficient planning algorithm for constrained motions," in *Advances in Robot Kinematics: Motion in Man and Machine*, J. Lenarcic and M. M. Stanisic, Eds. Springer Netherlands, pp. 251–260.

[12] J. Porta and L. Jaillet, "Path planning on manifolds using randomized higher-dimensional continuation," in *Algorithmic Foundations of Robotics IX*, ser. Springer Tracts in Advanced Robotics, D. Hsu, V. Isler, J.-C. Latombe, and M. Lin, Eds. Springer Berlin / Heidelberg, vol. 68, pp. 337–353.