Learning pose constraints for trajectory optimization from demonstration

Dominique M. Tao[1], Baris Akgun[2], Andrea L. Thomaz[2] and Aude Billard[1]

[1]Learning Algorithms and Systems Laboratory
EPFL, Switzerland
{dominique.tao,aude.billard}@epfl.ch

[2]Socially Intelligent Machine Lab
Department of Computer Science
UT Austin, USA
barisakgun@gmail.com
athomaz@ece.utexas.edu

***Abstract*— In the context of Keyframe-based Learning from Demonstration applied to robot manipulator, we present a framework for generating trajectories with a sampling-based motion planning by using the principal direction of the constraints in the learned model. Our approach first constructs a geometric constraint of the space between two gaussians, that we call Soft-Envelope, to constrain the movement of the robot. Dimension reduction is used on the Soft-Envelope to extract important aspects from the learned skill. Then we used our RRT* sampling-based motion planner to generate a trajectory w.r.t to the Soft-Envelope by using a double sampling approach combined with a natural gradient descent projection and jacobian in order to interleave between C-space and workspace. The tree grows in C-space by only using samples inside a subset corresponding to the Soft-Envelope in workspace.**

Kinesthetic teachning is used to collect data from a real 6-DOF Kinova Jaco arm. The performance of our algorithm is then tested with the same robot in simulation. During the execution, new unseen obstacles are introduced to hinder the movement.

***Index Terms*— keyframe-based Learning from Demonstration approach (KLfD), sampling-based motion planning, geometric constraints (Soft-Envelope), dimension reduction, natural gradient descent projection.**

I. INTRODUCTION

As robots are slowly becoming more and more present in our life, one of the ultimate goal will be to have them being able to automatically and autonomously help a person, if not completely replace him, in his every daily tasks. While it is very tedious and complex to pre-program a specific skill for a given task that can adapt several scenarios, the Learning from Demonstration (LfD) paradigm is an widely used alternative that enables the robot to learn and generalize a skill to reproduce by observing the performance of an user. In our framework, we focus on improving the creation and execution of a skill that the robot learned by using data collected from kinesthetic teaching, a common approach in LfD in which the user provides demonstrations by physically guiding the robot when performing a task as shown in figure 1. The nature of the demonstrations that will be used consist of keyframes, which are a sparse set of sequential points in space[2] [5][3]. A model of the skill is then constructed by clustering the keyframes in a statistical way, which can be seen as a set of constraints (position and orientation) that are used to execute the desired skill when connected in a correct sequential order.

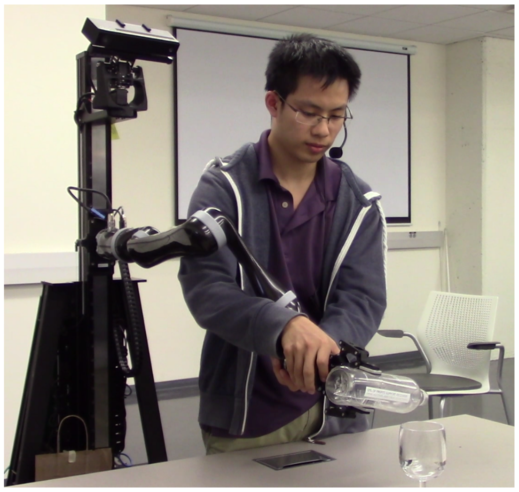

Fig. 1. A teacher using kinesthetic teaching with a 6-DOF Kinova Jaco arm to demonstrate a pouring water skill

Although using a spline to link the clusters together is a viable option in order to reproduce the skill [2], it may not be really robust when coping with a changing environment in which new obstacles that were not seen during the teaching could hinder the execution. One attempt to solve this issue was proposed by Andrey Kurenkov *et al.* in [7], in which he introduced an interactive GUI interface with the concept of Constrained-Keyframe (C-Keyframes), enabling the possibility for the user to view, edit or correct the skill model, and hence enabling a more efficient way of using a motion planning algorithm to deal with new obstacles and also having a more robust and coherent execution w.r.t the task. However, modifying the model skill for a new constrained scenario also means that it requires the intervention of the user.

In this work, we are interested in having the robot create a more coherent motion w.r.t to its task while being able to autonomously cope with new constrained environments. Our goal is then twofold: 1) to be able to avoid, without the intervention of the user, new static obstacles that were unseen during the teaching while satisfying the model constraints 2) making the execution more coherent with the demonstrated task, ie by first trying to move and avoid obstacles along the direction that is the most relevant to the task before trying the others.

An illustrative example could be a scenario where the robot was given the task to fill a glass with a bottle of water that it is holding in its end-effector. Naturally, in order to succeed the task, the bottle must end up above it but it can move more freely in the vertical direction. Hence, if an object was to obstruct the execution midway, the most relevant direction to try first in order to avoid it would be the vertical axis w.r.t to the glass before trying the others depending on the height of the obstacle.

If demonstrations were correctly performed by the user in this perspective, clusters of the model could clearly display the information of the direction with the biggest variance. However, as shown in previous work [5] [3], end-users tend to focus more on completing (*what to do*) the task rather than providing good demonstrations (*how to do*).

In order to encapsulate these different directions, we will push the analysis and dependence with the skill model further with geometric constraints and dimension reduction technique. A sampling-base motion planner will be used to deal with obstacles.

A. Proposed Approach

The approach that we propose in this paper can essentially be divided in three parts: learning, constrained manifold creation plus possibly dimension reduction and motion planning.

After collecting keyframes demonstrations with kinesthetic teaching, a skill model is created by clustering keyframes data in workspace. The clusters, in this work, are multivariate gaussians which can specify position and orientation constraints for the end-effector.

While the keyframes clusters are efficient to encapsulate the essence of the skill, there is no information on how the motion should be executed between two clusters, as they are sparsely distributed in space. To solve this issue and also for the used of the incoming motion planning algorithm, we constructed a "Soft-Envelope" between the two considered gaussians, named so because it designs an area in which we expect the motion of the end-effector to be. Dimension reduction method could be applied on the Soft-Envelope, resulting in a "Reduced Soft-Envelope" which will make the planner focus more on the direction with the most variance along the Soft-Envelope in order to execute the skill.

The creation of the trajectory is then handled by our sampling-based motion planner, which is an extended version of the RRT* planner. Our motivation to use such an algorithm lies in its abilities to efficiently find a feasible and collision-free path by growing a tree in space with an iterative sampling and without the user intervention. Furthermore, RRT* is probabilistically complete and asymptotically optimal if the sampling and the growth of the tree are done in the configuration space (C-space) of the robot, meaning that the planner is guaranteed to find a solution, if it exists, and that is optimal as the number of samples approaches infinity. As we are interested in the completeness and optimality of RRT*, our planner uses a double sampling process in order to explore the C-space while trying to have its tree growing in the Soft-Envelope. Natural gradient descent and jacobian are used to switch between both spaces, enabling the creation of a series of projected points that we leverage to construct the tree in C-Space while trying to satisfy the constraints defined by the Soft-Envelope in workspace.

B. Related Works

While regression technique such as GMM-GMR (gaussian mixture regression) proposed by Calinon *et al.* or spline interpolation in [2] by Baris *et al.* are used to generate trajectories from a encoded skill model, they often require the obstacles to be present in both the teaching and execution of the task. One alternative to deal with obstacles is to use sampling-based motion planners to create an end-to-end trajectory such as the RRT or RRT*, which were respectively introduced by LaValle [21] and Karaman [20] and created a lot of literature applied to robotics.

Shkolnik *et al.* proposed in [27] to grow a tree in task-space by direct sampling in this same space. Though it can achieve fast planning in very high dimension, the algorithm may not be complete because the grow of the tree is only done in workspace, ignoring the collision checking in C-space during the process.

Berenson *et al.* have done a lot of work, that have a lot in common with ours, with a single and (mainly) double complete RRT tree(s). In [15], they introduced the concept of workspace goal region (WGR) to specify goal end-effector poses that are sampled inside. Then they used an IK solver to grow their double tree IKBiRRT in C-Space. Difficulties of the IK and Jacobian methods were also pointed out by Bertram *et al.* in [11] who bypassed them by proposing a RRT that integrates the IK solution directly in the planner by adopting workspace heuristic functions that implicitly define goal regions of the configuration. In our work, we decided to use the jacobian instead of the IK solver, allowing our algorithm to be generalized for manipulator with more than 6 DOF where the IK solver might have encountered problems with infinite solutions for a given pose. Berenson *et al.* also work on a planner using projection technique such as in [9] in which he projects a joint angles on a constrained manifold in C-space or with the GradientT-RRT in [16], which searches toward lower cost regions to explore. The proof of completeness of RRT-based planner using sample-project method with Jacobian is given in [8]. Task Space Regions, along with workspace goal regions, are often used in order to represent the pose constraints for the projection in his sample-project method [10]. An other variation of projection method used with an RRT planner was also proposed by Mike Stilman in [28] with its RGD(random gradient descent)-RRT. Unlike their methods which are mostly projection of the joint configuration onto some constrained manifold, we are instead using a projection onto our constrained area in workspace with a double sampling and jacobian to guide the joint configuration in C-space. In addition, even though our constrained area can be related to their concept of Task-Space or Workspace goal region, the Soft-Envelope is more close to the representation of a prolate hyperspheroid area described in [18] proposed by Gammell *and al.* used to constrain sampling or to the Canal Space in [1] proposed by Chernova *et al.*mused to encode demonstrations in order to learn a skill model.

Work similar as ours in the sense that it combines Learning from demonstration ideas with sampling-based planner was done by Jonathan Classens in [13], in which he used a RRT with a model learned with GMM from trajectory data collected with kinesthetic teaching. The Gaussians of the model are used for direct sampling in workspace and an IK solver is used to grow the Tree in Cspace. Bowen *et al.* in [12] also proposed a method using PRM, an equivalent planner as RRT, with a model skill encoded by clusters motion features which are a similar concept as the keyframes clusters.

Another body of work have been done combining PCA dimension reduction method with a sampling-based planner to bias the RRT planner toward directions which it would not normally explore [22] or to favor the direction along which the roadmap tree is high [14]. PCA is also used with PRM planner to influence the sampling into difficult regions with low clearance in C-space [25]. In our work, we use eigenvalues/vectors-decomposition to influence sampling and the RRT tree growth along the principal component of each Gaussian used to create our constrained manifold in workspace.

II. Method

After collecting keyframes data (joint configuration and its corresponding pose) from kinesthetic teaching, the model is learned by using a HMM which gives as an output an order sequence of multivariate Gaussians to reproduce the skill as explained in [6]. At each time, our method is used on only two connected Gaussians from the given state sequence to construct a local trajectory, going respectively from what we called the "parent" to the "child" Gaussian. The full trajectory used to recreate the task can easily be generated by combining all the local trajectories together. The different parts of the algorithm are summarized in the pseudo-code of algorithm 1.

Let $\boldsymbol{q} \in \mathbb{R}^{M\times 1}$ a joint configuration in C-Space of dimension M and $\boldsymbol{p} \in \mathbb{R}^{N\times 1}$ a pose in workspace of dimension N. $\boldsymbol{\Sigma}_p, \boldsymbol{\Sigma}_c \in \mathbb{R}^{N\times N}$ and $\boldsymbol{\mu}_p, \boldsymbol{\mu}_c \in \mathbb{R}^{N\times 1}$ are respectively the covariance and center of the parent and child gaussians. The parameter α is used as the percentage of the data to be represented in the dimension reduction technique. The joint vector $\boldsymbol{q}_{start}$, with $\boldsymbol{p}_{start}$ its corresponding pose in workspace, is either the initial configuration of the robot or the last joint of the local trajectory solution from the previous pair of gaussians.

Algorithm 1 Planning($\boldsymbol{\Sigma}_p, \boldsymbol{\mu}_p, \boldsymbol{\Sigma}_c, \boldsymbol{\mu}_c, \alpha$)

```
output: Local trajectory from parent to children gaussian
1 $[Q_q, Q_p] \leftarrow \boldsymbol{q}_{start}, \boldsymbol{p}_{start}$;
2 $E \leftarrow \emptyset$;
3 $X_{sol} \leftarrow \emptyset$;
4 $G = (Q_q, Q_p, E, Q_{sol})$;
5 $[\boldsymbol{\Sigma}_{envSet}, \boldsymbol{\mu}_{envSet}] \leftarrow$ Soft_Envelope($\boldsymbol{\Sigma}_p, \boldsymbol{\mu}_p, \boldsymbol{\Sigma}_c, \boldsymbol{\mu}_c$);
6 $\boldsymbol{\Sigma}_{redEnvSet} \leftarrow$ Reduced_Soft_Envelope($\boldsymbol{\Sigma}_{envSet}, \boldsymbol{\Sigma}_p, \boldsymbol{\Sigma}_c, \alpha$);
7 PCTC_RRTStar($G, \boldsymbol{\Sigma}_{envSet}, \boldsymbol{\Sigma}_{redEnvSet}, \boldsymbol{\mu}_{envSet}, \alpha$);
```

For our variation of the RRT* algorithm, we used $T = (Q, E, Q_{sol})$ with $Q \subset Q_{free}$ the set of all vertices in C-space, the edges $E \subset Q_{free} \times Q_{free}$ and Q_{sol} the set of joints configuration inside the child gaussian, used to indicate that a path is found. The set Q is a double set Q_q and Q_p which respectively store the joint configuration used to grow the tree in C-space and also its corresponding pose in workspace.

A. Calibration

1) Child and Parent Gaussian:

For a given initial pose $\boldsymbol{p}_{init}$ of the end-effector, the first step is to know which pair of connected Gaussians to consider as parent and child for our planner.

We calculate the euclidean distance from $\boldsymbol{p}_{init}$ to each Gaussians center of the given state sequence and then select the Gaussian G^s that gives the minimal distance. Since G^s is from a sequence of ordered Gaussians, it is at most connected to a previous G^{s-1} and next Gaussian G^{s+1}. The euclidean distances from the pose p_{init} to the center of G^{s-1} and G^{s+1}, $\left\|\overrightarrow{\boldsymbol{p}_{init}\boldsymbol{\mu}^{s-1}}\right\|$ and $\left\|\overrightarrow{\boldsymbol{p}_{init}\boldsymbol{\mu}^{s+1}}\right\|$, are then calculated and compared to see which gaussian is to be paired with G^s for the first used of algorithm 1.

If $\left\|\overrightarrow{\boldsymbol{p}_{init}\boldsymbol{\mu}^{s-1}}\right\|$ is smaller, then G^{s-1} becomes the parent and G^s the child Gaussian (the same logic with G^s becoming the parent and G^{s+1} the child Gaussian if $\left\|\overrightarrow{\boldsymbol{p}_{init}\boldsymbol{\mu}^{s+1}}\right\|$ was smaller).

2) Soft-Envelope:

In a keyframe-based learning from demonstration, the skill is only encoded by a sparse set of keyframes Gaussians in workspace and hence there is no telling how the end-effector should move between two clusters, especially when using a sampling-based motion planner which constructs a trajectory by sampling the space.

We introduce a constrained area between two considered Gaussians with the intention of having a region inside which we would like the motion of the end-effector to be, which is why we called it "Soft-Envelope". A representation is shown

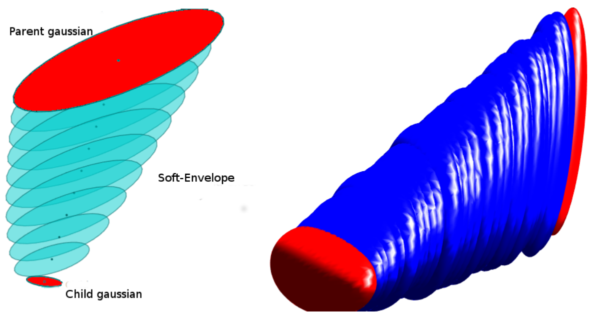

Fig. 2. Representation of the Soft-Envelope in 2D and 3D

in figure 2.
It is also used as a sampling region to guide the creation of the trajectory. In fact, only sampling inside the "child" Gaussian, depending on its size, may not be enough to efficiently grow an end-to-end path in presence of obstacles and sampling in the full C-space of the arm may lead the trajectory away from its goal.
The Soft-Envelope is created by linear interpolation of the parent/child covariance matrices and centers as shown in the following equations:

$$\begin{cases} \boldsymbol{\mu}_{env}{}^{0} = \boldsymbol{\mu}_p & \text{(1a)} \\ \boldsymbol{\Sigma}_{env}{}^{0} = \boldsymbol{\Sigma}_p & \text{(1b)} \\ \boldsymbol{\mu}_{env}{}^{i+1} = \boldsymbol{\mu}_{env}{}^{i} + (\boldsymbol{\mu}_c - \boldsymbol{\mu}_p)\triangle d & \text{(1c)} \\ \boldsymbol{\Sigma}_{env}{}^{i+1} = \boldsymbol{\Sigma}_{env}{}^{i} + (\boldsymbol{\Sigma}_c - \boldsymbol{\Sigma}_p)\triangle d & \text{(1d)} \end{cases}$$

with $\boldsymbol{\mu}_{env}{}^{i} \in \mathbb{R}^{N\times 1}$ and $\boldsymbol{\Sigma}_{env}{}^{i} \in \mathbb{R}^{N\times N}$ the center and covariance matrix of the i^{th} gaussian of the Soft-Envelope. We define $\boldsymbol{\Sigma}_{envSet}$ the set of all the calculated matrices $\boldsymbol{\Sigma}_{env}$ forming the Soft-Envelope and $\boldsymbol{\mu}_{envSet}$ the set of all $\boldsymbol{\mu}_{env}{}^{i}$. Though the coefficient $\triangle d$ can be chosen as a result of a function with respect to the euclidean distance of the two center $\left\|\overrightarrow{\boldsymbol{\mu}_p\boldsymbol{\mu}_c}\right\|$, we decided instead to use a small number (w.r.t. the biggest and smallest distance between two linked gaussians in the model) since not all pair of gaussians in the model are evenly separated.
Furthermore, the Soft-Envelope can automatically be adapted to any pair of Gaussians and has the advantage of being easily an uniformly sampled since it is only composed of a series of Gaussians (see section II-B.2).

3) Reduced Soft-Envelope:

As the variance of a Gaussian in the model can be interpreted as the intention of the user when the skill was demonstrated, the Soft-Envelope is also used as a way to influence the end-effector in the direction it should first try to move, w.r.t to the variance of its Gaussians, in order avoid any obstacles. For this goal, the idea is to encapsulate the direction with the biggest variances in each of its Gaussians by using a eigenvalues/vectors decomposition technique, creating a "Reduced-Soft-Envelope" in full workspace that we used instead of the Soft-Envelope.
As a fact, one has first to remember that only the parent and the child Gaussians of the model truly represent the variance allowed for the task, the Soft-Envelope being just one linear interpretation of the variations should evolve between these two. Hence, our first step will be to use eigenvalue decomposition separately on the child and parent Gaussians to determine the number of eigenvector needed to represent $\alpha\%$ of the data, with α a parameter specified by the user. We compare them and take the biggest number of eigenvector as N_{eig}.
Each of the Soft-Envelope Gaussians covariance matrix $\boldsymbol{\Sigma}^{i}_{env} \in \mathbb{R}^{N\times N}$ are then decomposed into a full eigenvector matrix $\boldsymbol{V}^{i} \in \mathbb{R}^{N\times N}$ and diagonal eigenvalue matrix $\boldsymbol{D}^{i} \in \mathbb{R}^{N\times N}$. The reduced matrix $\boldsymbol{V}^{i}_{red} \in \mathbb{R}^{N\times N_{eig}}$ and $\boldsymbol{D}^{i}_{red} \in \mathbb{R}^{N_{eig}\times N_{eig}}$ are created by using the N_{eig} biggest eigenvectors and associated eigenvalues.
The full reduced covariance matrix from $\boldsymbol{\Sigma}^{i}_{env}$ is finally calculated with $\boldsymbol{\Sigma}^{i}_{red} = \boldsymbol{V}^{i}_{red}\boldsymbol{D}^{i}_{red}(\boldsymbol{V}^{i}_{red})^{\dagger}$, such that $\boldsymbol{\Sigma}^{i}_{red} \in \mathbb{R}^{N\times N}$ is the covariance matrix of the i^{th} Gaussian of the Reduced Soft-Envelope. Each new reduced covariance matrix belongs to the set $\boldsymbol{\Sigma}_{redEnvSet}$. A visual representation is shown in figure 3.

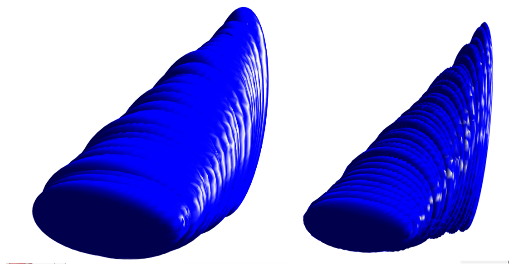

Fig. 3. Representation of (left image)the Soft-Envelope ($\alpha = 100$) and (right image) Reduced-Soft-Envelope($\alpha = 90$)

One particular aspect to be cautious with the reduced Soft-Envelope is to make sure that its Gaussians are still semi-positive definite because they will be used later to calculate a quadratic distance (Mahalanobis distance). In fact, $\boldsymbol{\Sigma}^{i}_{red}$ being calculated by approximation of eigenvectors and eigenvalues matrices can have approximation of very small values making it just not positive definite.
One way to correct this problem is to use the method introduced by Higham in [19] to calculate the nearest symmetric semi-definite positive matrix that we can combine with a Tikhonov regularization using the smallest eigenvalue until the matrix is positive definite as shown in algorithm 2.

Algorithm 2 Tikhonov regularization

```
k=0 and $\varepsilon = FLT\_EPSILON$;
while $\lambda_{Smallest} < 0$ do
    k++;
    $\boldsymbol{\Sigma}_{red}{}^{i} = \boldsymbol{\Sigma}_{red}{}^{i} + (-\lambda_{Smallest} \times k^2 + \varepsilon) \times \mathbf{I}$;
end
```

*B. PCTC-RRT**

Our sampling-based planner is an extension of the RRT*. The work has been more focus on how the sampling, w.r.t the (reduced) Soft-Envelope, should be consider in order to

influence the trajectory (see subsection II-B.1). The work was more focus on how the sampling needed to be considered w.r.t to the Soft-Envelope. The pseudo-code of the procedure is presented in the algorithm 3.
Our objective is to grow a single Tree in C-space while trying to be constrained inside the Soft-Envelope. For that purpose, we use a double sampling, one sample in each space, that we combined with a natural gradient descent projection. The first sampling $\boldsymbol{p}_{sRef}$ (line 3) is done inside the Soft-Envelope in Workspace. It is then used as the reference sampling pose on which $\boldsymbol{p}_s^j$ is projected, where $\boldsymbol{p}_s^j$ is the corresponding pose of the second sample $\boldsymbol{q}_s$ (line 7-8).

During the projection process(line 10-33), preprocessing steps with the natural gradient is used to iteratively calculate the next pose $\boldsymbol{p}_s^{j+1}$ toward $\boldsymbol{p}_{sRef}$ (line 15-21). We then feed-forward the previous pose $\boldsymbol{p}_s^j$ to have a first error $\boldsymbol{e1} = \boldsymbol{p}_s^{j+1} - \boldsymbol{p}_s^j$ and also use the pose of the end-effector calculated from the previous natural gradient update, $\boldsymbol{p}_{sEef}$ (line 9 or 27), as a feedback to have a second error $\boldsymbol{e2} = \boldsymbol{p}_s^{j+1} - \boldsymbol{p}_{sEef}$ (line 22-24). The sum of them, $\boldsymbol{e} = \boldsymbol{e1} + \boldsymbol{e2}$, is used with the Jacobian, $\boldsymbol{J}$, which is calculated with the joint configuration $\boldsymbol{q}_s$ in order to update this same joint configuration with the discrete Euler formula. Forward kinematic is then used to find the next pose of the end-effector $\boldsymbol{p}_{sEef}$(line 25-27). When $\boldsymbol{p}_s$ is projected on $\boldsymbol{p}_{sRef}$, we use the new updated joint angles $\boldsymbol{q}_s$ to grow the RRT* tree (line 29-32).A representation of the process can be visualized with a block diagram in figure 4.

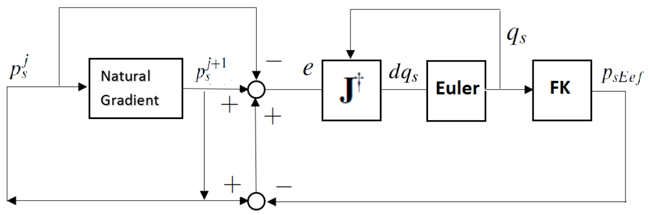

Fig. 4. Block diagram of the projection process

In more details, the feed-forward error $\boldsymbol{e1}$ updates the joint configuration $\boldsymbol{q}_s$ along the direction of the natural gradient projection. In the same logic, the feedback error $\boldsymbol{e2}$ indicates along which direction $\boldsymbol{q}_s$ should be updated to diminish the error between its pose and the pose calculated with the natural gradient. The evaluation of the algorithm is also possible without the feedback, but a small offset error will be created between $\boldsymbol{p}_{sEef}$ and $\boldsymbol{p}_s^{j+1}$ over the projection iterations since only the direction is feed-forwarded to $\boldsymbol{q}_s$.
In other words, we solve the IK problem for every sample $\boldsymbol{p}_{sRef}$ in the Soft-Envelope by using a natural gradient descent projection of poses in workspace in order to guide the second sample $\boldsymbol{q}_s$ such that we find a joint configuration for $\boldsymbol{p}_{sRef}$. This way of solving the IK problem, despite being a bit slow due to all these projections, allow us to have different joints configuration for a same point and thus properly explore the C-Space.
Furthermore, while we project the pose onto the reference inside the Soft-Envelope in workspace, we also exploit the series of created joints configuration $\boldsymbol{q}_s$ whose poses $\boldsymbol{p}_s^j$ are inside the Soft-Envelope to grow the RRT* tree (line 11-14).
`Get_Membership_SoftEnv()` evaluates which gaussian is the closest to the pose $\boldsymbol{p}_s^j$ and if it is inside the Soft-Envelope. It calculates the mahalanobis distance from $\boldsymbol{p}_s^j$ to each of its i^{th} gaussians and then select the gaussian with the smallest distance. Then this gaussian is used to check if $\boldsymbol{p}_s^j$ is inside the Soft-Enveloep with $(\boldsymbol{p}_s^j - \boldsymbol{\mu}_{env}^i)^\top (\boldsymbol{\Sigma}_{env}^{select})^{-1}((\boldsymbol{p}_s^j - \boldsymbol{\mu}_{env}^i)) < criteria$. The value for the criteria is detailed in [17] (however more strict values can be use). In case we have $\alpha < 100$, then we still used the covariance of the Soft-Envelope matrices to find the gaussian with the smallest mahalanobis distance but we use $(\boldsymbol{\Sigma}_{redEnv}^{select})$ instead of $(\boldsymbol{\Sigma}_{env}^{select})$ to check its belonging to the Reduced Soft-Envelope

1) *The core of PCTC-RRT*:*

While the work was more focus in the sampling in PCTC-RRT*, the core of the algorithm remains almost the same as a RRT*. The algorithm 4 presents the pseudo-code of a RRT* with, in red, the changes that we made.
New vertices and edges are added to $G = (Q_q, Q_p, E, Q_{sol})$ by growing the tree in the free C-space towards random selected states. Then a "choose parent" and "rewiring" steps are done for each new vertex such that it minimizes the cost of nearby vertices in the Tree (more details can be found in [20]). In addition, every time a new joint configuration node is added to Q_q, we calculate its pose with forward kinematic which is added in the Q_p set.
The change that was made in `Nearest`$(G, \boldsymbol{q}, \boldsymbol{p})$ (line 1) is that it finds the closest node $\boldsymbol{p}_{Nearest}$ from the set Q_p to the pose $\boldsymbol{p}$ by using the mahalanobis distance with the child covariance matrix $\boldsymbol{\Sigma}_c$ in workspace. Since the two set Q_q and Q_p have the same number of node and ordered in the same way, we also get $\boldsymbol{q}_{Nearest}$ which will be used to grow the RRT* tree.
We also changed the condition to tell when a path is found (line 22-27). When a new node $\boldsymbol{q}_{New}$ in C-space is added to the Tree, we use its corresponding pose $\boldsymbol{p}_{New}$ (found with forward kinematic and added to Q_p) to calculate the mahalanobis distance to the children gaussian and evaluate its belonging. If we have $(\boldsymbol{p}_{New} - \boldsymbol{\mu}_c^i)^\top (\boldsymbol{\Sigma}_c^i)^{-1}((\boldsymbol{p}_{New} - \boldsymbol{\mu}_c^i)) < criteria$ (with the value for criteria as discuted just above), then a solution path from the parent to the child gaussian is found (line 25). We can either directly update the next pair of parent/child gaussian in order to plan for the next new local trajectory or continue the sampling to find a more optimal path until the allowed time is reached or any other criterias specified by the user is met (local biasing can also be added to algorithm 5 once a path is found as suggested by Baris *and al.* in [4] in order to focus the sampling more around the best solution path)

2) *Sampling:*

The sampling that was done inside the Soft-Envelope follows the procedure illustrated by the pseudo-code in algorithm 5. We use a goal biasing heuristic to sample inside the child gaussian, which can drastically change the planning speed (line 2-4). In that case, the covariance matrix used to generate samples inside the child gaussian is just $\boldsymbol{\Sigma}_c$. Otherwise we sample uniformly inside the Soft-Envelope (line 4-9). First, we randomly select a point on the axis connecting the center of the parent and child gaussian. This pose, $\boldsymbol{p}_{sel}$, is then used in the function `Gauss_Lin_Interpol(`$\boldsymbol{p}_{sel}$`)` to calculate the covariance matrix $\boldsymbol{\Sigma}_{sel}$ of the envelope gaussian with $\boldsymbol{p}_{sel}$ as its center. In fact, `Gauss_Lin_Interpol(`$\boldsymbol{p}_{sel}$`)` performs the same linear interpolation of the covariance matrix as in equation 1d with $\Delta d = \|\overrightarrow{\boldsymbol{\mu}_p\boldsymbol{\mu}_{sel}}\| / \|\overrightarrow{\boldsymbol{\mu}_p\boldsymbol{\mu}_c}\|$, with $\boldsymbol{\mu}_{sel} = \boldsymbol{p}_{sel}$. Limitations of the estimation of the calculation is also set by using the unit vector $\overrightarrow{\boldsymbol{U}_{\boldsymbol{\mu}_p\boldsymbol{\mu}_c}} = \overrightarrow{\boldsymbol{\mu}_p\boldsymbol{\mu}_c} / \|\overrightarrow{\boldsymbol{\mu}_p\boldsymbol{\mu}_c}\|$: in case we have $\overrightarrow{\boldsymbol{U}_{\boldsymbol{\mu}_{sel}\boldsymbol{\mu}_c}} \neq \overrightarrow{\boldsymbol{U}_{\boldsymbol{\mu}_p\boldsymbol{\mu}_c}}$, then it means that the center μ_{sel} is not in-between the two considered gaussians, ie it is outside from the perspective of the child gaussian, thus the covariance and center are set such as $\boldsymbol{\Sigma}_{sel} = \boldsymbol{\Sigma}_c$ and $\boldsymbol{\mu}_{sel} = \boldsymbol{\mu}_c$; in case of $\overrightarrow{U_{\boldsymbol{\mu}_p\boldsymbol{\mu}_{sel}}} \neq \overrightarrow{U_{\boldsymbol{\mu}_p\boldsymbol{\mu}_c}}$, then the center μ_{sel} is outside from the perspective of the parent gaussian, thus we have $\boldsymbol{\Sigma}_{sel} = \boldsymbol{\Sigma}_p$ and $\boldsymbol{\mu}_{sel} = \boldsymbol{\mu}_p$.
Once we have $\boldsymbol{\Sigma}_{sel}, \boldsymbol{\mu}_{sel}$, we then generate one sample from this gaussian. In case a dimension reduction need to be perform ($\alpha < 100$), we simply reduce the gaussian $\boldsymbol{\Sigma}_{sel}$ as explained in the section II-A.3 with the algorithm **??** line (8-18) before generating the sample.

3) Projection - Natural gradient descent:

Because we are using a mahalanobis distance w.r.t to the gaussians of the Soft-Envelope instead of an euclidean distance to update the pose $\boldsymbol{p}_s^j$ for the projection, the natural gradient descent is used [26] to minimize the projection cost. The cost of the gradient takes into account two influences: the gaussian that the reference pose $\boldsymbol{p}_{sRef}$ belongs to (that we are calling goal gaussian in this subsection) and the Soft-Envelope itself. While using only the goal gaussian is possible for the projection, integrating the influence of the Soft-Envelope will make the valley of the gradient cost function bend toward it. In other words, the line created by the projected points will be curved towards the Soft-Envelope. This behavior is desirable to speed the growth of the Tree since every joint configuration whose pose is inside the envelope is used to grow the RRT* tree (algorithm 3 line 12-14).
The goal gaussian is determined with the function `Get_Membership_SoftEnv(`$\boldsymbol{p}_{sRef}$`)` which output the covariance matrix $\boldsymbol{\Sigma}_{sRef}$ of the goal gaussian with $\boldsymbol{p}_{sRef}$, used as the center of the goal gaussian.(algorithm 3 line 6)
The influence of the Soft-Envelope is taken into account by first projecting the pose $\boldsymbol{p}_s^j$ onto the axis created by the center $\boldsymbol{\mu}_p, \boldsymbol{\mu}_c$ of the parent and child gaussians with the function `Axis_SoftEnv_Ortho_Projection(`$\boldsymbol{p}_s^j$`)`, using the formula $\boldsymbol{p}_{proj} = \boldsymbol{\mu}_p + \boldsymbol{p}_{tmpProj}$ with $\boldsymbol{p}_{tmpProj} = \frac{(\overrightarrow{\boldsymbol{\mu}_p\boldsymbol{p}_s^j} \cdot \overrightarrow{\boldsymbol{\mu}_p\boldsymbol{\mu}_c})}{\|\overrightarrow{\boldsymbol{\mu}_p\boldsymbol{\mu}_c}\|^2} \overrightarrow{\boldsymbol{\mu}_p\boldsymbol{\mu}_c}$
The projected point $\boldsymbol{p}_{proj}$ on the axis is then used in the function `Gauss_Lin_Interpol(`$\boldsymbol{p}_{proj}$`)` to find the center $\boldsymbol{\mu}_g^j$ and covariance $\boldsymbol{\Sigma}_g^j$ of the gaussian inside the Soft-Envelope that will influence the gradient cost function (algorithm 3 line 15-16).

One has to notice that $\boldsymbol{p}_{sRef}$ which is not necessarily on the axis of the parent-child center was chosen as the center for $\boldsymbol{\Sigma}_g$, creating an offset in the projection. In order to correct it, the same orthogonal projection function is used on the reference sample $\boldsymbol{p}_{sRef}$ to calculate this "offset" which will be added to $\boldsymbol{\mu}_g^j$ before it will be used in the natural gradient update(algorithm 3 line 4-5 and 17).

The natural gradient is calculated by using the mahalanobis metric such as:

$$coeff = \frac{\|\overrightarrow{\boldsymbol{\mu}_c\boldsymbol{\mu}_p}\|}{(\boldsymbol{\mu}_p - \boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}_c^{-1} (\boldsymbol{\mu}_p - \boldsymbol{\mu}_c)} \tag{2}$$

$$\begin{cases} F_{Goal} = (\boldsymbol{p}_s^j - \boldsymbol{p}_{sRef})^\top \boldsymbol{\Sigma}_{sRef}^{-1} (\boldsymbol{p}_s^j - \boldsymbol{p}_{sRef}) & \text{(3a)} \\ F_{Env} = (\boldsymbol{p}_s^j - \boldsymbol{p}_{proj}^j)^\top \boldsymbol{\Sigma}_g^{-1} (\boldsymbol{p}_s^j - \boldsymbol{p}_{proj}^j) & \text{(3b)} \end{cases}$$

$$\begin{cases} \boldsymbol{dF}_{Goal} = 2\boldsymbol{\Sigma}_{sRef}^{-1} (\boldsymbol{p}_s^j - \boldsymbol{p}_{sRef}) & \text{(4a)} \\ \boldsymbol{dF}_{Env} = 2\boldsymbol{\Sigma}_g^{-1} (\boldsymbol{p}_s^j - \boldsymbol{p}_{proj}^j) & \text{(4b)} \end{cases}$$

The update of p_s^j is done with :

$$\boldsymbol{p}_s^j = \boldsymbol{p}_s^j + \delta_n(\boldsymbol{\Sigma}_g(-coeff \times \boldsymbol{dF}_{Env}) + \boldsymbol{\Sigma}_{sRef}(-coeff \times \boldsymbol{dF}_{Goal})) \tag{5}$$

with δ_n the natural gradient stepsize.
The choice of the stepsize is important for a correct convergence of the projection. Numerous techniques exist to calculate the stepsize of a gradient descent projection, but most of them still create too much points because of the value of the stepsize which is chosen small to avoid divergence. In addition of being slow, this might also cause an unwanted biasing of the tree growth toward some specific regions other than the child gaussian.
Since the goal of the projection is different for each new sample of $\boldsymbol{p}_{sRef}$, we use an adaptive step-size for each new projection by using the same formula illustrated in [23] used in a reinforcement learning framework, p. 289 equation 6.

$$\begin{cases} step1 = (coeff \times \boldsymbol{dF}_{Goal})^\top \boldsymbol{\Sigma}_{sRef} (coeff \times \boldsymbol{dF}_{Goal}) & \text{(6a)} \\ step2 = (coeff \times \boldsymbol{dF}_{Env})^\top \boldsymbol{\Sigma}_g (coeff \times \boldsymbol{dF}_{Env}) & \text{(6b)} \\ \delta_n = \frac{1}{\sqrt{(step1 + step2)}} & \text{(6c)} \end{cases}$$

The calculation of δ_n is only done once at the beginning of each new projection (ie for each new sample $\boldsymbol{p}_{sRef}$), which allows a faster projection as the poses at the beginning are more distant. In fact, if the next pose updated by the natural gradient converges towards $\boldsymbol{p}_{sRef}$, then the rest of the projection will also converge for the same calculated

step-size because the mahalanobis distance from each new updated pose to $\boldsymbol{p}_{sRef}$ is decreasing.
Since the context of application of this formula is different, we implemented an additional step to avoid divergence of the projection. If the mahalanobis distance of the first updated pose $\boldsymbol{p}_s^{j+1}$ is bigger than the distance of the original pose sample $\boldsymbol{p}_s^j$, then it means that the step of the new updated pose was too big and then the projection will diverge. We corrected this problem by diminishing the step-size N times until convergence ($\delta_n = \delta_n \times a$ with $0 < a < 1$, in a similar way as the step-size line search backtracking method), with N a parameter defined by the user. If after N times, the criteria is still not met, then the projection is not evaluated.

Algorithm 3 PCTC_RRTStar(G,$\boldsymbol{\Sigma}_{envSet}$,$\boldsymbol{\Sigma}_{redEnvSet}$,$\boldsymbol{\mu}_{envSet}$, α)

input : In addition, $\boldsymbol{\Sigma}_p, \boldsymbol{\Sigma}_c, \boldsymbol{\mu}_p$ and $\boldsymbol{\mu}_c$ are assumed available for all the functions here
output: Solution path connecting the parent and child gaussian

```
//First loop: creation of workspace and C-space samples,
while TRUE or ALLOWED TIME do
    $\boldsymbol{p}_{sRef} \leftarrow$ SoftEnv_Sampling($\boldsymbol{\Sigma}_{envSet}, \boldsymbol{\Sigma}_{redEnvSet}, \alpha$);
    $\boldsymbol{p}_{projRef} \leftarrow$ Axis_SoftEnv_Ortho_Projection($\boldsymbol{p}_{sRef}$);
    $\boldsymbol{v}_{offset} = \boldsymbol{p}_{sRef} - \boldsymbol{p}_{projRef}$;
    $[\boldsymbol{\Sigma}_{sRef}, flag_{sRef}] \leftarrow$
    Get_Membership_SoftEnv($\boldsymbol{p}_{sRef}, \boldsymbol{\Sigma}_{envSet}, \boldsymbol{\Sigma}_{redEnvSet}, \boldsymbol{\mu}_{envSet}$);
    $\boldsymbol{q}_s \leftarrow$Random_Real_From0To2PI();
    $\boldsymbol{p}_s^j \leftarrow$ Forward_Kin($\boldsymbol{q}_s$);
    $\boldsymbol{p}_{sEef} = \boldsymbol{p}_s^j$;
    //Second loop: Gradient projection and Tree growing
    while TRUE or ALLOWED TIME do
        $[\boldsymbol{\Sigma}_2, flag2_{belong}] \leftarrow$
        Get_Membership_SoftEnv($\boldsymbol{p}_s^j, \boldsymbol{\Sigma}_{envSet}, \boldsymbol{\Sigma}_{redEnvSet}, \boldsymbol{\mu}_{envSet}$);
        if $flag2_{belong} == TRUE$ then
            $S_{path} \leftarrow$ Extended_RRTStar($G, \boldsymbol{q}_s, \boldsymbol{p}_s$);
        end
        $\boldsymbol{p}_{proj} \leftarrow$ Axis_SoftEnv_Ortho_Projection($\boldsymbol{p}_s^j$);
        $[\boldsymbol{\Sigma}_g, \boldsymbol{\mu}_g] \leftarrow$ Gauss_Lin_Interpol($\boldsymbol{p}_{proj}$);
        $\boldsymbol{\mu}_g = \boldsymbol{\mu}_g + \boldsymbol{v}_{offset}$;
        $[\boldsymbol{p}_s^{j+1}, flag_{diverge}] \leftarrow$
        Natural_Gradient($\boldsymbol{p}_s^j, \boldsymbol{\Sigma}_{sRef}, \boldsymbol{p}_{sRef}, \boldsymbol{\Sigma}_g, \boldsymbol{\mu}_g$);
        if $flag_{diverge} == TRUE$ then
            break;
        end
        $\boldsymbol{e}_1 = \boldsymbol{p}_s^{j+1} - \boldsymbol{p}_s^j$;
        $\boldsymbol{e}_2 = \boldsymbol{p}_s^{j+1} - \boldsymbol{p}_{sEef}$;
        $\boldsymbol{e} = \boldsymbol{e}_1 + \boldsymbol{e}_2$;
        $\boldsymbol{J} \leftarrow$GetJacobian($\boldsymbol{q}_s$);
        $\boldsymbol{q}_s = \boldsymbol{q}_s + \boldsymbol{J}^\dagger \boldsymbol{e}$;
        $\boldsymbol{p}_{sEef} \leftarrow$ Forward_Kin($\boldsymbol{q}_s$);
        $\boldsymbol{p}_s^j = \boldsymbol{p}_s^{j+1}$
        if $\boldsymbol{p}_s$ projected on $\boldsymbol{p}_{sRef}$ then
            $G \leftarrow$ Extended_RRTStar($G, \boldsymbol{q}_s, \boldsymbol{p}_{sRef}$);
            break;
        end
    end
end
```

Algorithm 4 Extended_RRTStar($G, \boldsymbol{q}, \boldsymbol{p}$)

```
output: set of vertices and edges $G = (Q_q, Q_p, E, Q_{sol})$
1  $[\boldsymbol{q}_{Nearest}, \boldsymbol{p}_{Nearest}] \leftarrow$ Nearest($G, \boldsymbol{q}, \boldsymbol{p}$);
2  $\boldsymbol{q}_{New} \leftarrow$ Steer($\boldsymbol{q}_{Nearest}, \boldsymbol{q}$);
3  if Collision_Free($q_{Nearest}, \boldsymbol{q}_{New}$) then
4      $Q_q \leftarrow Q_q \cup \boldsymbol{q}_{New}$;
5      $\boldsymbol{q}_{Min} \leftarrow \boldsymbol{q}_{Nearest}$;
6      $Q_{Near} \leftarrow$ Near($G, \boldsymbol{q}$);
7      forall the $\boldsymbol{q}_{Near} \in Q_{Near}$ do
8          if Collision_Free($\boldsymbol{q}_{Near}, \boldsymbol{q}_{New}$) then
9              if Cost($\boldsymbol{q}_{Near}$)+CostIm($\boldsymbol{q}_{Near}, \boldsymbol{q}_{New}$)<Cost($\boldsymbol{q}_{New}$) then
10                 $\boldsymbol{q}_{Min} \leftarrow Q_{Near}$;
11             end
12         end
13     end
14     $E \leftarrow E \cup (\boldsymbol{q}_{Min}, \boldsymbol{q}_{New})$;
15     forall the $\boldsymbol{q}_{Near} \in Q_{Near} \backslash \boldsymbol{q}_{Min}$ do
16         if Collision_Free($\boldsymbol{q}_{Near}, \boldsymbol{q}_{New}$) and Cost($\boldsymbol{q}_{New}$)+CostIm($\boldsymbol{q}_{Near}, \boldsymbol{q}_{New}$) < Cost($\boldsymbol{q}_{Near}$) then
17             $\boldsymbol{q}_{Par} \leftarrow$ Parent($\boldsymbol{q}_{Near}$);
18             $E \leftarrow E \setminus (\boldsymbol{q}_{Par}, \boldsymbol{q}_{Near})$;
19             $E \leftarrow E \cup (\boldsymbol{q}_{New}, \boldsymbol{q}_{Near})$;
20         end
21     end
22     $\boldsymbol{p}_{New} \leftarrow$ Forward_Kin($\boldsymbol{q}_{New}$);
23     $Q_p \leftarrow Q_p \cup \boldsymbol{p}_{New}$;
24     dist_goal $\leftarrow$ MahalanobisDist($\boldsymbol{p}_{New}, \boldsymbol{\Sigma}_c, \boldsymbol{\mu}_c$);
25     if dist_goal < GaussBelongCrit then
26         $Q_{sol} \leftarrow Q_{sol} \cup \boldsymbol{p}_{New}$;
27     end
28 end
```

Algorithm 5 SoftEnv_Sampling($\boldsymbol{\Sigma}_{envSet}, \boldsymbol{\Sigma}_{redEnvSet}, \alpha$)

```
input : In addition, $\boldsymbol{\Sigma}_p, \boldsymbol{\Sigma}_c, \boldsymbol{\mu}_p$ and $\boldsymbol{\mu}_c$ are assumed available
        for all the functions here
output: Sample $\boldsymbol{q}_s$ in C-space
1  randProb ←Random_Real_From0To2PI();
2  if randProb < GoalBias then
3      $\boldsymbol{\Sigma}_{sel} = \boldsymbol{\Sigma}_c$;
4  else
5      randNum ←Random_Real_From0To2PI();
6      $\boldsymbol{U}_{axis} = (\boldsymbol{\mu}_c - \boldsymbol{\mu}_p)/\overrightarrow{\boldsymbol{\mu}_p\boldsymbol{\mu}_c}$;
7      $\boldsymbol{p}_{sel} = \boldsymbol{\mu}_p + \boldsymbol{U}_{axis} * (randNum \times \overrightarrow{\boldsymbol{\mu}_p\boldsymbol{\mu}_c})$;
8      $[\boldsymbol{\Sigma}_{sel}, \boldsymbol{\mu}_{sel}] \leftarrow$ Gauss_Lin_Interpol($\boldsymbol{p}_{sel}$);
9  end
10 if $\alpha$ < 100 then
11     $\boldsymbol{\Sigma}_{redSel} \leftarrow$ reduction $\boldsymbol{\Sigma}_{sel}$ as in algorithm 2 (line 8-18);
12     $\boldsymbol{q}_s \leftarrow$ Sample_Gaussian($\boldsymbol{\Sigma}_{redSel}$);
13 else
14     $\boldsymbol{q}_s \leftarrow$ Sample_Gaussian($\boldsymbol{\Sigma}_{sel}$);
15 end
```

III. Experiments

We first use kinesthetic teaching with a real 6-dof Kinova Jaco arm mounted on a robot platform (figure 1) to collect keyframes data and a simulated robot in simulation to emulate its behavior and obtain results.
We then evaluated our algorithm with Moveit-Rviz simulator. The program is used as an input for our algorithm written in C++. The behavior of the robot arm was tested with 2 different tasks. The setup for each of them was with a 0.61m x 1.22m x 0.73m table on which were disposed different objects from the YCB set. Simulation was done on a Lenovo computer with a Intel Core i7-5600U CPU 2.60GHz $\times$ 4 processor and Nvidia GeForce 940M graphic card.
The initial configuration of the arm is placed above the table with its end-effector inside the first gaussian. We tested our algorithm with 2 simple tasks whose skill models, learned with 7 successful demonstrations, are only encoded with two gaussians. Moreover, in the next figures, the parent gaussian is in red and the child one is in blue. For each task, the algorithm was run 7 times, with dimension reduction of $\alpha = 85\%$ and without it.
Our objective is to show how our algorithm with dimension reduction influences the generation of an end-to-end trajectory w.r.t to the Soft-Envelope while finding the correct direction to avoid the obstacle. For the two tasks, we stop the planner once a first path is found, better optimal solutions could have been found by letting the algorithm run longer.

IV. Results

A. Task 1: Pick and drop

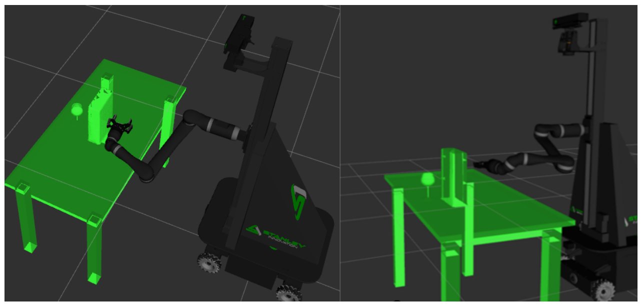
Fig. 5. Setup of the task 1: a plastic wine glass and a Cheez-it box on the table

The task is to have the end-effector, that is holding a small object(cherry fruit), go above a wine glass and release its gripper to drop the object inside. For the motion planning, a Cheez-It box is used to obstruct the arm movement as shown in figure 5. The skill of the model in figure 6 shows that the end-effector has a lot of variance in the z direction above the glass. In this task, our objective is to show how our algorithm with dimension reduction influences the trajectory generation even though the direction with the most variance is already clearly display by the gaussian. Results of the generated trajectories are pictured in figure 6
We can clearly see that all the trajectories with dimension reduction first follow the variance of the first gaussian

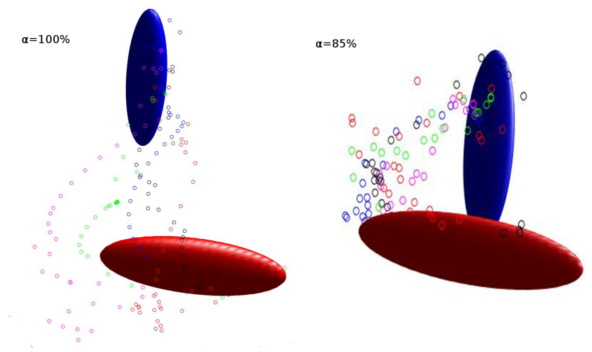

Fig. 6. Plot of 5 different trajectories (distinction by color) in x,y,z space for task 1 with (right image) $\alpha = 85\%$ and (left image)$\alpha = 100\%$

(avoiding the obstacle by its side) before going up (certainly following the variance of the second one), which shows the influence of our algorithm. While the trajectories don't look as optimal as the one calculated without reduction, they are more coherent w.r.t the demonstration given by the user (and in addition, given the same situation (example of pouring water into a glass), most people probably would have avoided the large obstacle the same way).

B. Task 2: Pick and place

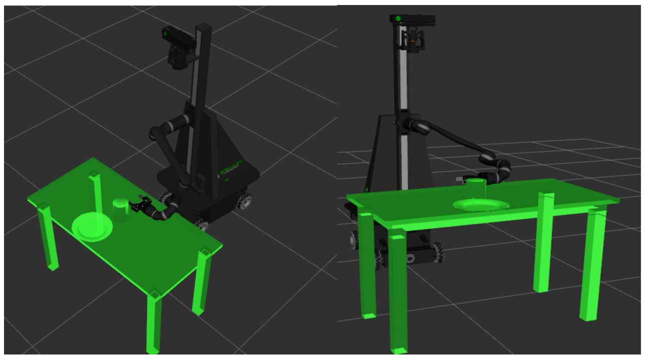

Fig. 7. Setup of task 2: a plate and a coffee can on the table

The setup in task 2 is pictured in figure 7 in which we used the coffee can in the middle to hinder the motion. Unlike the model of task 1, this one, displayed in figure 8, show gaussians whose principal directions are less visible. The objective in this task is to show how that, with less obvious gaussians (could be from user focusing on completing the task instead of giving good demonstrations), our algorithm will still encapsulate and use these directions to influence the creation of trajectory. We can clearly see that without dimension reduction, the trajectories are more messy, in particular the blue and black one who are starting going left but completely deviated to the right. Furthermore, one small detail to be noticed about the influence of our algorithm with dimension reduction: trajectories start more by going along the principal component of the first gaussian unlike the other case.

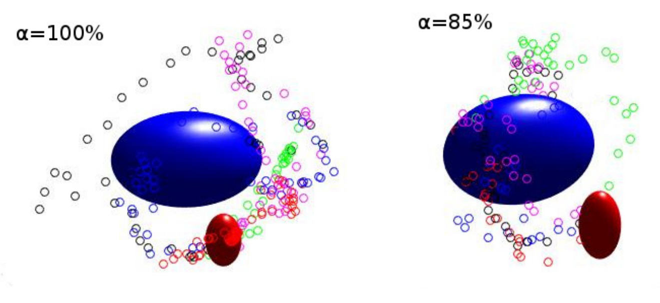

Fig. 8. Plot of 5 different trajectories (distinction by color) in x,y,z space for task 2 with (right image) $\alpha = 85\%$ and (left image)$\alpha = 100\%$

C. Tasks Performance

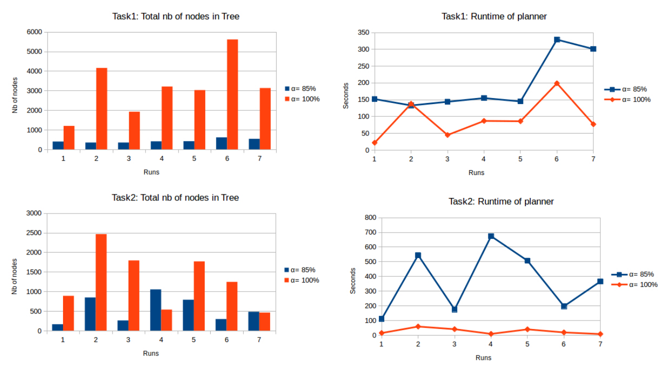

Fig. 9. Plots: total number of node in tree (col. 1) and Runtime(col. 2) for the first (row 1) and second (row 2) task

All the trajectory displayed in the previous subsections were able to achieve the task, however the performances with and without dimension reduction are different, as shown with the runtime and total number of node in the tree in the figure 9 for the first and second task. As expected in this regards, both task display similar performance results. It takes less time without reduction techniques because more joints configuration, whose poses are inside the full Soft-Envelope, are used to grow the tree than with the reduced Soft-Envelope. Indeed, the projected poses belongs more easily to the Soft-Envelope than to the reduced one since its gaussians are bigger, which is why the total number of nodes in the tree is in general bigger than the one with reduce dimension.

V. Future Work

In this work, we have presented an approach to generate trajectory with sampling-based planner which is more dependent on the skill model to find the correct direction to avoid obstacles. However, the model may not be fully representative of the direction to take in order to avoid obstacles depending on their form. The logical next step would be to integrate the environment, encapsulate the shapes and variances of the objects with a kinect for instance, with the variance of the model to calculate more precisely the best direction to take in order to avoid obstacles. An other direction of work could be to further modify the algorithm such that it could plan a trajectory with dynamic obstacles (as it is done by suggested by Otte *et al.* [24] with its RRTx)

VI. Conclusion

We have presented a trajectory planner combined with a learned skill from keyframes demonstration. The concept of Soft-Envelope was proposed to constrain the area between two keyframes gaussians in which an extension of the RRT*, PCTC-RRT*, was used to find a non-collision and available configuration space path linking two considered gaussians whose correspondent poses were around/inside the Soft-Envelope. The planner interleaves the exploration of C-space while being constrained in an area in workspace by using a double sampling process combined with gradient projection and jacobian. The algorithm was successfully tested with two different tasks, showing the possible influences of the model to make the motion more coherent to the task.

VII. Acknowledgements

The authors would like to thank Aude Billard for making this joint research possible, Andrea L. Thomaz and Akgun Baris for the valuable time that they spent for help and discussions.

References

[1] S. Reza Ahmadzadeh and Sonia Chernova. Encoding demonstrations and learning new trajectories using canal surfaces. In *25th International joint Conference on Artificial Intelligence (IJCAI 2016), Workshop on Interactive Machine Learning: Connecting Humans and Machines*, pages 1–7, 2016.

[2] Baris Akgun, Maya Cakmak, Karl Jiang, and Andrea Lockerd Thomaz. Keyframe-based learning from demonstration - method and evaluation. *I. J. Social Robotics*, 4(4):343–355, 2012.

[3] Baris Akgun, Maya Cakmak, Jae Wook Yoo, and Andrea Lockerd Thomaz. Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. pages 391–398, 2012.

[4] Baris Akgun and Mike Stilman. Sampling heuristics for optimal motion planning in high dimensions. In *IROS*, pages 2640–2645. IEEE, 2011.

[5] Baris Akgun and Andrea Lockerd Thomaz. Learning constraints with keyframes.

[6] Baris Akgun and Andrea Lockerd Thomaz. Self-improvement of learned action models with learned goal models. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*, pages 5259–5264, 2015.

[7] Baris Andrey Kurenkov, Akgun and Andrea Lockerd Thomaz. An evaluation of gui and kinesthetic teaching methods for constrained-keyframe skills. September 2015.

[8] Dmitry Berenson and Siddhartha Srinivasa. Probabilistically complete planning with end-effector pose constraints. In *IEEE International Conference on Robotics and Automation (ICRA '10)*, May 2010.

[9] Dmitry Berenson, Siddhartha Srinivasa, David Ferguson, and James Kuffner. Manipulation planning on constraint manifolds. In *IEEE International Conference on Robotics and Automation (ICRA '09)*, May 2009.

[10] Dmitry Berenson, Siddhartha Srinivasa, and James Kuffner. Task space regions: A framework for pose-constrained manipulation planning. *International Journal of Robotics Research (IJRR)*, 30(12):1435 – 1460, October 2011.

[11] Dominik Bertram, James Kuffner, Ruediger Dillmann, and Tamim Asfour. An integrated approach to inverse kinematics and path planning for redundant manipulators. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1874–1879. IEEE, May 2006.

[12] Gu Ye Chris Bowen and Ron Alterovitz. Asymptotically optimal motion planning for learned tasks using time-dependent cost maps. *IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, VOL. 12, NO. 1,*, JANUARY 2015.

[13] Jonathan Claassens. An rrt-based path planner for use in trajectory imitation. In *ICRA*, pages 3090–3095. IEEE, 2010.

[14] S. Dalibard and J.P. Laumond. Linear dimensionality reduction in random motion planning. *International Journal of Robotics Research*, 30(12):pp. 1461–1476, 2011.

[15] Dave Ferguson Alvaro Collet James J. Kuffner Dmitry Berenson, Siddhartha Srinivasa. Manipulation planning with workspace goal regions.

[16] Thierry Simeon Dmitry Berenson and Siddhartha Srinivasa. Addressing cost-space chasms in manipulation planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) , 4561-4568.*, May 2011.

[17] P. Filzmoser. A multivariate outlier detection method. In *State University*, pages 18–22, 2004.

[18] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Informed rrt*: Optimal incremental path planning focused through an admissible ellipsoidal heuristic. *CoRR*, abs/1404.2334, 2014.

[19] Nicholas J. Higham. Computing a nearest symmetric positive semidefinite matrix.

[20] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *CoRR*, abs/1005.0416, 2010.

[21] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[22] Y. Li and K. E. Bekris. Balancing state-space coverage in planning with dynamics. In *IEEE International Conference on Robotics and Automation (ICRA10)*, Anchorage, AK, May 2010 2010.

[23] Takamitsu Matsubara, Tetsuro Morimura, and Jun Morimoto. Adaptive step-size policy gradients with average reward metric. In Masashi Sugiyama and Qiang Yang, editors, *ACML*, volume 13 of *JMLR Proceedings*, pages 285–298. JMLR.org, 2010.

[24] Michael Otte and Emilio Frazzoli. RRT-X: Asymptotically optimal single-query sampling-based motion planning with quick replanning. *The International Journal of Robotics Research*, 2015.

[25] Jan Rosell, Raúl Suárez, and Alexander Pérez. Path planning for grasping operations using an adaptive pca-based sampling method. *Auton. Robots*, 35(1):27–36, 2013.

[26] S.Amari and S.C. Douglas. Why natural gradient ?

[27] Alexander C. Shkolnik and Russ Tedrake. Path planning in 1000+ dimensions using a task-space voronoi bias. In *ICRA*, pages 2061–2067. IEEE, 2009.

[28] Mike Stilman. Navigation among movable obstacles. October 2007.