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*— When performing a task in presence of obstacles, there are multiple directions the manipulator could take to avoid them and still successfully execute the task. In the context of Keyframe-based Learning from Demonstration applied to robot manipulator, we present a framework with a sampling-based planner that generates a trajectory based on the intention of the user which indicates towards which direction the robot should first try to move in order to avoid obstacles while executing the task. Our approach first constructs a geometric constraint of the space between two Gaussians of the model, 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. It uses a double sampling approach combined with a natural gradient descent projection and Jacobian transform in order explore the C-space while being constrained around the Soft-Envelope. The tree is grown in C-space by only using samples inside a C-space subset corresponding to the Soft-Envelope in workspace.**
Kinesthetic teachning is used to collect keyframes data from a real 6-DOF Kinova Jaco arm. 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

Kinesthetic teaching in a Learning from Demonstration context is a widely used approach that allows an user to easily provide demonstrations to the robot by physically guiding it when performing a skill as shown in figure 1. However, when executing the learned skill from demonstrations, the robot may encounter new and unseen obstacles that were not present during teaching while executing the skill. Different actions and directions can be undertook by the robot to avoid them and still complete the task.
If we take the pouring water skill illustrated in figure 1 as an example and we insert an obstacle mid-way, one possible action is to have the user re-demonstrate the skill to take them into account, and thus indicating one possible direction among many possible others to avoid the obstacle. It is often the case when using a regression technique (GMM-GMR) [9] [8] or spline interpolation [3] to create the skill, the obstacle has to be present in both teaching and execution or new demonstrations need to be provided to take them into account. Still in the context of regression technique, reinforcement learning is an action that can be undertaken to enable the learning of new solution without the intervention of the user[16]. An other course of action is to use a motion planner such as a potential field [1] or a sampling based-planner(RRT [11], PRM[10]), combined with the learned model, to create an end-to-end trajectory for the execution of the task without the intervention of the user, but the obstacle avoidance is done more randomly. Although these methods successfully accomplish the task, they only select one or random directions for the robot to avoid an obstacle and sometimes require the intervention of the user.

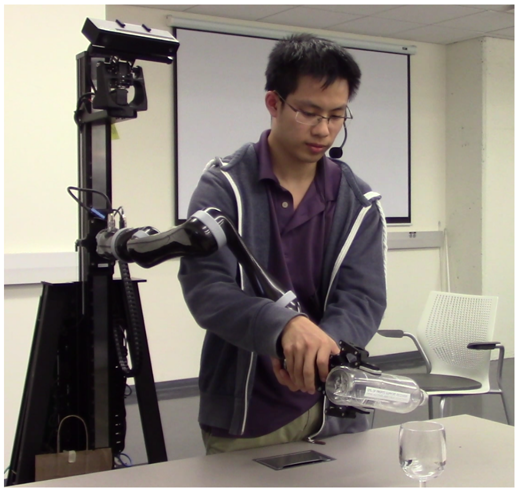
Fig. 1. A teacher using kinesthetic teaching with a 6-DOF Kinova Jaco arm to demonstrate a pouring water skill

Our objective in this framework is then two-fold: 1) to enable the robot to avoid, without the help of the user, new static obstacles during its execution 2) indicate to the robot a first direction (which we assimilate as the user intention when he demonstrated the skill to the robot) to try in order to avoid obstacles before it tries other directions.

In this paper, we use keyframes demonstration provided by kinesthetic teaching to learn the skill. Keyframes are a sparse set of sequential points or critical points in space. They are also referred as *via-points* in industrial robot programming. During the demonstrations, the user moves the robot to the desired pose and records a keyframes. The resulting skill model is composed of multivariate Gaussians that are clustering these keyframes.
We also made the assumption that the biggest variance of each Gaussian of the model reflects the intention of the user when demonstrating the skill. This variance is identified as the first direction that the robot should try to move to avoid an obstacle. For instance, with the pouring water skill, we expect the demonstration of the Gaussian above the glass to have more variance in its vertical direction since there is less restriction from where the water is spill.

A. Proposed Approach

To satisfy the above-mentioned objectives, we combine the learned skill model which specify constraints in space (position and orientation for the end-effector to perform the task) with the performance of a RRT* sampling-based planner, introduced by Karaman [18].
Our motivation to use such an planner lies in its abilities to efficiently find a feasible and collision-free path that randomly explore any direction 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 (done uniformly) and the growth of the tree are 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.
The planner that we proposed in this work is an extension of a RRT* that was designed to efficiently sample inside a constrained manifold. In fact, 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 our planner, we constructed a constrained area between two considered Gaussians, that we named "Soft-Envelope".
In order to influence the motion w.r.t the intention of the user, dimension reduction method is applied on the Soft-Envelope, resulting in a "Reduced Soft-Envelope". If correctly reduced, it enables our planner to direct its trajectory along the some principal directions by constraining its sampling inside the reduced region that reflects the biggest variance along the Soft-Envelope.
Even though there is no proof of it in this paper, we are still interested in satisfying the criteria of completeness and optimality for RRT* as just mentioned above. Hence our planner uses a double sampling process in order to uniformly explore the C-space while trying to have its tree growing inside the Soft-Envelope. Natural gradient descent projection and Jacobian transform are also used for this reason.
Moreover, the Jacobian transform is used instead of an IK solver to find the joint configuration, 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.

B. Related Works

Berenson *et al.* have done many works in sampling-based planning and constrained manifold that can relate to ours. He introduces the concept of Workspace Goal Region [13] or Task Space Regions(Chains) [7] which is a 6-dimensional volume in workspace specifying the goal sets for their planner. Although these constraints are intuitive to specify, can efficiently be sampled and the distance from a joint configuration to one of this constraint can be easily calculated, they need to be manually set depending on the task. In our work, the multivariate Gaussians from the skill model can be assimilated to the Task Space Regions, sharing all their advantages, but is instead automatically defined by the user when he provided the demonstration. Our Soft-Envelope constraints can be assimilated to the concept of Task Space Region Chains but also do not need to be manually defined as it follows naturally from the Gaussian clusters of the skill model. In addition, a similar representation of our Soft-Envelope constraint can be found in [2] with the Canal Space proposed by Ahmadzadeh *et al.* who uses it to encode demonstrations in order to learn a skill model.
Berenson *et al.* also work on a planners using projection technique [13] [14] by projecting the joint configuration onto their constrained manifold (proof of completeness is done in [6]). Similarly, our planner also use projection technique with a gradient descent but instead project one pose in workspace onto another one inside the Soft-Envelope because it is more easy to express the distance w.r.t to a gaussian in workspace. The projection is used as a feed-forward to guide the joint configuration inside a subspace of C-space corresponding to the Soft-Envelope in Workspace.

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 [20] or to favor the direction along which the roadmap tree is high [12]. PCA is also used with PRM planner to influence the sampling into difficult regions with low clearance in C-space [23]. In our work, we use eigenvalues/vectors-decomposition to influence sampling and the growth of the tree along the principal component of each Gaussian used to create our constrained region 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 ordered sequence of multivariate Gaussians in workspace to reproduce the skill. A more detailed explanation of the procedure is given in [5]. At each time, our method is used on only two connected Gaussians from the given states 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 initialize the tree with $T = (Q, E, Q_{sol})$ 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:

We introduce a constrained area between two considered Gaussians of the model with the intention of having a constrained region inside which we would like the motion of the end-effector to be, which is why we called it "Soft-Envelope". An illustration is shown in figure 2.

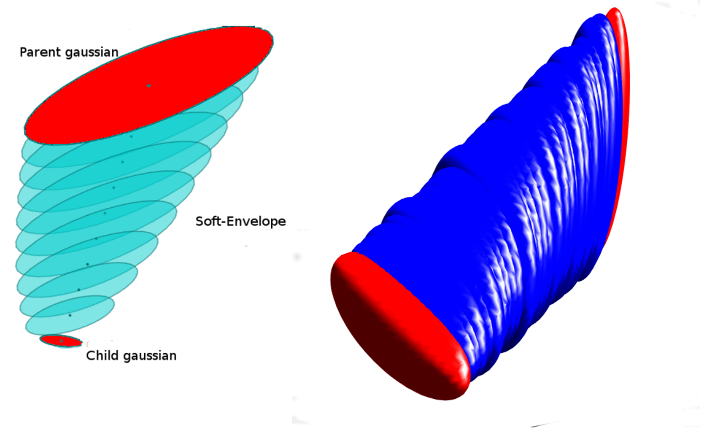

Fig. 2. Representation of the Soft-Envelope in 2D and 3D ($\mathbb{R}^3$)

In fact, 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 of the model (parent and child), especially when using a sampling-based motion planner which constructs a trajectory by sampling the space.
It is also used as a sampling region to guide the creation of the trajectory. 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 the goal, with the consequences of waiting a long time before convergence.

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 sampled in a uniform way 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 Gaussians of the Soft-Envelope are then viewed as a linear transition of the user intention. They are hence used as a way to influence (indicate to) the end-effector which directions it should first try to move, when going from the parent to the child Gaussian, in order avoid obstacles.

The idea to do so is to encapsulate the direction with the biggest variances in each of its Gaussians by using an eigenvalues/vectors decomposition technique, creating a "Reduced-Soft-Envelope" in full workspace that we used, instead of the Soft-Envelope, for the sampling and for the evaluation of any point belongings to it. The process is illustrated in algorithm 2.

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 how the variances should evolve between these two. Hence, our first step uses eigenvalues 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} (line 2-5).

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 (line 8-10). Each of this new covariance matrix belongs to the set $\boldsymbol{\Sigma}_{redEnvSet}$. A visual representation is shown in figure 3.

Algorithm 2 Reduced_Soft_Envelope($\boldsymbol{\Sigma}_{envSet}$,$\boldsymbol{\Sigma}_p$,$\boldsymbol{\Sigma}_c$, α)

```
output: All $\boldsymbol{\Sigma}_{red}{}^{i}$ of the Reduced Soft-Envelope ($\Sigma_{redEnvSet}$ the set of all $\boldsymbol{\Sigma}_{red}{}^{i}$)
1  //Number of eigenvector $N_{eig}$ to represent $\alpha\%$ of the data
2  $N_{eigP} \leftarrow$ Extract_N_EigenDecomp($\boldsymbol{\Sigma}_p$, $\alpha$);
3  $N_{eigC} \leftarrow$ Extract_N_EigenDecomp($\boldsymbol{\Sigma}_c$, $\alpha$);
4  if $N_{eigC} < N_{eigP}$ then $N_{eig} = N_{eigP}$;
5  else $N_{eig} = N_{eigC}$;
6  //Reduced Soft-Envelope
7  for $i \leftarrow 1$ to size($\boldsymbol{\Sigma}_{envSet}$) do
8      $[\boldsymbol{V}^i,\boldsymbol{D}^i] \leftarrow$ Eigen_Decomp($\boldsymbol{\Sigma}_{env}{}^{i}$);
9      $[\boldsymbol{V}^i_{red},\boldsymbol{D}^i_{red}] \leftarrow$ Reduce_Mat($\boldsymbol{V}^i$,$\boldsymbol{D}^i$,$N_{eig}$);
10     $\boldsymbol{\Sigma}_{red}{}^{i} = \boldsymbol{V}^i_{red}\boldsymbol{D}^i_{red}(\boldsymbol{V}^i_{red})^{\dagger}$;
11     //Calculate nearest positive definite matrix (Higmam method) and regularization
12     $\boldsymbol{\Sigma}_{red}{}^{i} \leftarrow$ Nearest_PDMat($\boldsymbol{\Sigma}_{red}{}^{i}$);
13     $EigVal_{Smallest} \leftarrow$ Eigen_Decomp($\boldsymbol{\Sigma}_{red}{}^{i}$);
14     k=0 and $\varepsilon = FLT\_EPSILON$;
15     while $EigVal_{Smallest} < 0$ do
16         k++;
17         $\boldsymbol{\Sigma}_{red}{}^{i}$= $\boldsymbol{\Sigma}_{red}{}^{i} + (-EigVal_{Smallest} \times k^2 + \varepsilon) \times \mathbf{I}$;
18     end
19 end
```

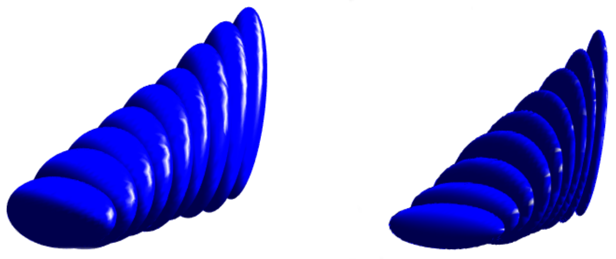
Fig. 3. Representation in ($\mathbb{R}^3$) of the same Soft-Envelope (left image)without dimension reduction ($\alpha = 100\%$) and (right image) with dimension reduction($\alpha = 90\%$ reduced Soft-Envelope); We can see the effect of the reduction that redefined the contour and make the Gaussian smaller w.r.t the selected eigenvectors.

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 [17] to calculate the nearest symmetric semi-definite positive matrix that we combine with a regularization process using the smallest eigenvalue until the matrix is positive definite (line 12-18).

B. PCT-RRT*

Our sampling-based planner (Projection Constrained Taskspace -RRT*) PCT-RRT* is an extension of the RRT*: the work has been more focus on how the sampling should be considered to grow a single tree in C-space while being influenced by the Soft-Envelope, however the core of the planner remains a RRT*. The pseudo-code of the procedure is presented in the algorithm 3.
As our interest lies in growing the tree inside the Soft-Envelope while being able to explore the C-space (one of the necessary condition for the completeness of the planner), we use, at each iteration, a double sampling process (line 3,7,8).

At each iteration, a first sample, $\boldsymbol{p}_{sRef}$, is created inside the Soft-Envelope in workspace and its joint configuration is used to grow the tree in C-space. To generate its IK solution, we use a second sampling, $\boldsymbol{q}_s$, in the full C-space, and iteratively project, in a second intern loop, its corresponding pose $\boldsymbol{p}_s^j$ onto $\boldsymbol{p}_{sRef}$ by using a gradient descent process combined with the Jacobian transform.
During the projection process (line 10-33), the natural gradient descent is used to iteratively calculate the next pose $\boldsymbol{p}_s^{j+1}$ from $\boldsymbol{p}_s^j$ toward $\boldsymbol{p}_{sRef}$ (line 18). We then use its previous pose $\boldsymbol{p}_s^j$ to calculate the feed-forward error $\boldsymbol{e}_1$ and the pose of the end-effector from the previous joint configuration update $\boldsymbol{p}_{sEef}$ (line 9,27) to calculate a feedback error $\boldsymbol{e}_2$, which are used with the Jacobian transform to update the joint configuration $\boldsymbol{q}_s$ and its corresponding pose $\boldsymbol{p}_{sEef}$ (line 22-28). The error, used to update the next $\boldsymbol{q}_s$, is calculated such that it gives the direction of the natural gradient update towards $\boldsymbol{p}_{sRef}$ and also the direction to decrease the error between the natural gradient calculated poses and the end-effector one. When $\boldsymbol{p}_s$ is finally projected on $\boldsymbol{p}_{sRef}$, we use its new updated joint angles $\boldsymbol{q}_s$ to grow the RRT* tree (line 29-32).
In other words, even though $\boldsymbol{p}_s^j$ comes from the second sampling $\boldsymbol{q}_s$ only at the beginning of the projection, the rest of its update $\boldsymbol{p}_s^{j+1}$ is only due to the natural gradient projection towards $\boldsymbol{p}_{sRef}$; the direction of the natural gradient at each iteration is used to calculate the new joint configuration $\boldsymbol{q}_s$ whose corresponding pose $\boldsymbol{p}_{sEef}$ should be equal to $\boldsymbol{p}_s^{j+1}$ pose. A representation of the process can be visualized in figure 5 and with a block diagram in figure 4.

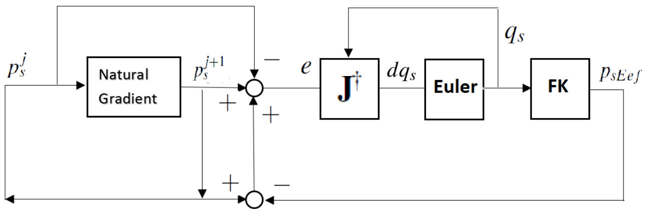

Fig. 4. Block diagram of the projection process

Despite being a bit slow due to all these projections, this way of solving the IK problem enables the algorithm to

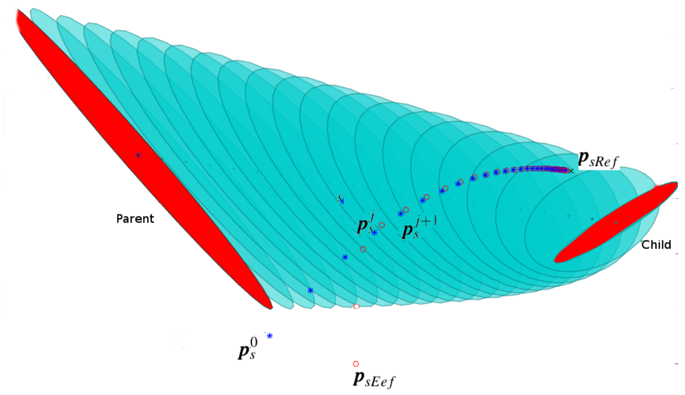

Fig. 5. Projection process of $\boldsymbol{p}_s^j$ onto $\boldsymbol{p}_{sRef}$ with natural gradient descent guiding $\boldsymbol{p}_{sEef}$, the pose of $\boldsymbol{q}_s$

have different joints configuration for a same point and thus properly explore the C-Space while constraining the sample inside the Soft-Envelope.
Furthermore, to accelerate the process, we also exploit, during the projection, the series of intermediate 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(`$\boldsymbol{p}_s^j$`)` evaluates which Gaussians of the Soft-Envelope is the closest to the pose $\boldsymbol{p}_s^j$ then if it belongs to it. It calculates the Mahalanobis distance from $\boldsymbol{p}_s^j$ to each of its i^{th} Gaussian and then select the one with the smallest distance. Then this Gaussian, with its covariance matrix $\boldsymbol{\Sigma}_{env}^{select}$ and center $\boldsymbol{\mu}_{env}^{select}$, is used to check if $\boldsymbol{p}_s^j$ is inside the Soft-Envelope with the Mahalanobis distance such that this condition $(\boldsymbol{p}_s^j - \boldsymbol{\mu}_{env}^{select})^\top(\boldsymbol{\Sigma}_{env}^{select})^{-1}((\boldsymbol{p}_s^j - \boldsymbol{\mu}_{env}^{select})) < criteria$ holds. The value for the criteria is detailed in [15] (however, in practice, more strict values can be use).
In case we have a dimension reduction ($\alpha < 100$), the Soft-Envelope is still used to find the gaussian with the smallest mahalanobis distance but its belonging is checked by using its corresponding Gaussian in the reduced Soft-Envelope.

1) The core of PCT-RRT*:

While the work is more focused on the sampling in PCT-RRT*, the core of the algorithm, represented by the function `Extended_RRTStar(`$G, \boldsymbol{q}_s, \boldsymbol{p}_s$`)` in algorithm 3 and whose process is illustrated in algorithm 4, remains almost the same as a RRT*.

Algorithm 3 PCT_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
    $p_{sRef} \leftarrow$ Sampling_SoftEnv($\Sigma_{envSet}, \Sigma_{redEnvSet}, \alpha$);
    $p_{projRef} \leftarrow$ Axis_SoftEnv_Ortho_Projection($p_{sRef}$);
    $v_{offset} = p_{sRef} - p_{projRef}$;
    $[\Sigma_{sRef}, flag_{sRef}] \leftarrow$ Get_Membership_SoftEnv($p_{sRef}, \Sigma_{envSet}, \Sigma_{redEnvSet}, \mu_{envSet}$);
    $q_s \leftarrow$Sampling_CSpace_0-2PI();
    $p_s^j \leftarrow$ Forward_Kin($q_s$);
    $p_{sEef} = p_s^j$;
    //Second loop: Gradient projection and Tree growing
    while TRUE or ALLOWED TIME do
        $[\Sigma_2, flag2_{belong}] \leftarrow$ Get_Membership_SoftEnv($p_s^j, \Sigma_{envSet}, \Sigma_{redEnvSet}, \mu_{envSet}$);
        if $flag2_{belong} == TRUE$ then
            $S_{path} \leftarrow$ Extended_RRTStar($G, q_s, p_s$);
        end
        $p_{proj} \leftarrow$ Axis_SoftEnv_Ortho_Projection($p_s^j$);
        $[\Sigma_g, \mu_g] \leftarrow$ Gauss_Lin_Interpol($p_{proj}$);
        $\mu_g = \mu_g + v_{offset}$;
        $[p_s^{j+1}, flag_{diverge}] \leftarrow$ Natural_Gradient($p_s^j, \Sigma_{sRef}, p_{sRef}, \Sigma_g, \mu_g$);
        if $flag_{diverge} == TRUE$ then
            break;
        end
        $e_1 = p_s^{j+1} - p_s^j$;
        $e_2 = p_s^{j+1} - p_{sEef}$;
        $e = e_1 + e_2$;
        $J \leftarrow$GetJacobian($q_s$);
        $q_s = q_s + J^\dagger e$;
        $p_{sEef} \leftarrow$ Forward_Kin($q_s$);
        $p_s^j = p_s^{j+1}$
        if $p_s$ projected on $p_{sRef}$ then
            $G \leftarrow$ Extended_RRTStar($G, q_s, p_{sRef}$);
            break;
        end
    end
end
```

For the process of a conventional RRT*, 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 $\boldsymbol{q}_s$: for a new sample node $\boldsymbol{q}_s$ given to the RRT*, the nearest node from the Q_q set is found in a euclidean distance metric. Then `Steer(`$\boldsymbol{q}_{Nearest}, \boldsymbol{q}_s, \delta_r$`)` generates a new node $\boldsymbol{q}_{New}$ which is δ_r away from the nearest node $\boldsymbol{q}_{Nearest}$ towards the direction of the sample node $\boldsymbol{q}_s$. In case there is no collision in the line from the nearest node to the new created one, checked with the function `Collision_Free(`$\boldsymbol{q}_{Nearest}, \boldsymbol{q}_{New}$`)` , then a "choose parent" and "rewiring" steps are done for each new node such that it minimizes the cost of nearby nodes in the tree that are within a certain euclidean distance to the new node, found with `Near(`$G, \boldsymbol{q}_{New}$`)` .
The "choose parent" step evaluates the cost of each node in the neighborhood of $\boldsymbol{q}_{New}$ from the root node and select the one that minimizes this cost to be the parents of $\boldsymbol{q}_{New}$. The step of "rewiring" that follows evaluates each cost from $\boldsymbol{q}_{New}$ to its neighbor node, to which is added the cost from the root node of the tree to $\boldsymbol{q}_{New}$, in order to select the neighbor node that has the minimum cost such that $\boldsymbol{q}_{New}$ is its parent (or to be the children of $\boldsymbol{q}_{New}$). `Cost(`$\boldsymbol{q}_{Near}$`)` calculates the cumulative cost from the root node of the tree to $\boldsymbol{q}_{Near}$ and `CostIm(`$\boldsymbol{q}_{Near}, \boldsymbol{q}_{New}$`)` calculates the direct cost of going from $\boldsymbol{q}_{Near}$ to $\boldsymbol{q}_{New}$ (a more detailed explanation of the RRT* process is given in [19] and [18]).

In our variation of the RRT* (all changes are in red in algorithm 4), we use an additional set Q_p which takes every pose found from forward kinematics of its corresponding joint configuration node added to Q_q.
An other change is made when the planner is looking for the nearest node to the sample one in `Nearest(`$G, \boldsymbol{q}_s, \boldsymbol{p}_s$`)`. Instead of searching in C-space, it finds the closest node $\boldsymbol{p}_{Nearest}$ in workspace from the Q_p set to the sample pose $\boldsymbol{p}_s$ by using the Mahalanobis distance with the child covariance matrix $\boldsymbol{\Sigma}_c$. Since the two set Q_q and Q_p have the same number of node and are ordered in the same way, we also get $\boldsymbol{q}_{Nearest}$ which is used to grow the RRT* tree. While there is not concrete proof, we observed that this modification allows the tree to be more constrained around the Soft-Envelope. In fact, instead of having the nearest joint configuration to the joint angles sample and thus growing a tree only based on joint angles without knowing anything about their poses, we decided to look for the nearest pose, which has more chance of being inside the Soft-Envelope to grow the tree in C-space with its joint angles. A comparison can be found in figure 6

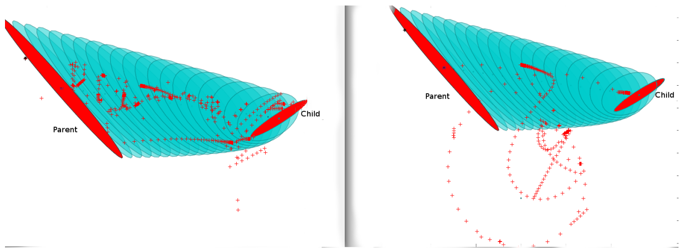

Fig. 6. All poses in the tree (red cross) with the Nearest function look for: (left) nearest pose, (right) nearest joints angles; The left one is more constrained around the Soft-Envelope

We also changed the condition to tell when a path is found by the planner (line 22-27). When a new node $\boldsymbol{q}_{New}$ in C-space is added to the Tree, we use its corresponding pose $\boldsymbol{p}_{New}$ to calculate the Mahalanobis distance to the child Gaussian and evaluate its belonging with $(\boldsymbol{p}_{New} - \boldsymbol{\mu}_c^i)^\top(\boldsymbol{\Sigma}_c)^{-1}((\boldsymbol{p}_{New} - \boldsymbol{\mu}_c^i)) < criteria$ (value for *criteria* mentioned in section II-B). If this condition holds, then a solution path from the parent to the child Gaussian is found. We can then 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 criteria specified by the user is met. Local biasing can also be added to the sampling in algorithm 5 once a path is found as suggested by Akgun *and al.* in [4] in order to focus the sampling more around the best solution path to improve it faster .

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

```
output: set of vertices and edges $G = (Q_q, Q_p, E, Q_{sol})$
1  $[\boldsymbol{q}_{Nearest}, \boldsymbol{p}_{Nearest}] \leftarrow$ Nearest($G, \boldsymbol{q}_s, \boldsymbol{p}_s$);
2  $\boldsymbol{q}_{New} \leftarrow$ Steer($\boldsymbol{q}_{Nearest}, \boldsymbol{q}_s, \delta_r$);
3  if Collision_Free($\boldsymbol{q}_{Nearest}, \boldsymbol{q}_{New}$) then
4      $Q_q \leftarrow Q_q \cup \boldsymbol{q}_{New}$;
5      $\boldsymbol{q}_{Min} \leftarrow \boldsymbol{q}_{Nearest}$;
6      $Q_{Near} \leftarrow$ Near($G, \boldsymbol{q}_{New}$);
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             $E \leftarrow E \ \backslash(\boldsymbol{q}_{Par}, \boldsymbol{q}_{Near})$;
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
```

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 either sample inside the child gaussian, which can drastically change the planning speed, or to uniformly sample inside the (reduced) Soft-Envelope (line 2-9).

When sampling inside the child Gaussian, the covariance matrix used to generate samples is just $\boldsymbol{\Sigma}_c$. Otherwise we sample uniformly inside the Soft-Envelope by first randomly selecting a pose, $\boldsymbol{p}_{sel}$,on the axis connecting the center of the parent and child Gaussians.

`Gauss_Lin_Interpol`($\boldsymbol{p}_{sel}$) is then used with this pose to calculate the covariance matrix $\boldsymbol{\Sigma}_{sel}$ and center $\boldsymbol{\mu}_{sel}$ of the envelope Gaussian. To do so, this function performs the same linear interpolation of the covariance matrix as in equation 1d with $\Delta d = \|\overrightarrow{\boldsymbol{\mu}_p\boldsymbol{\mu}_{sel}}\| / \|\overrightarrow{\boldsymbol{\mu}_p\boldsymbol{\mu}_c}\|$, and $\boldsymbol{\mu}_{sel} = \boldsymbol{p}_{sel}$. Limitations of the center $\boldsymbol{\mu}_{sel}$ 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 $\boldsymbol{\mu}_{sel}$ is not in-between the two considered parent/child 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 with $\boldsymbol{\Sigma}_{sel}\boldsymbol{r} + \boldsymbol{\mu}_{sel}$, with $\boldsymbol{r} \in \mathbb{R}^{M\times 1}$ a vector with M random values.

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 2 line (8-18) before using it to generate the sample.

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

```
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 $\leftarrow$Random_Real_From0To2PI();
2  if randProb < GoalBias then
3      $\boldsymbol{\Sigma}_{sel} = \boldsymbol{\Sigma}_c$;
4  else
5      randNum $\leftarrow$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
```

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 chosen [24] 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 makes the valley of the gradient cost function bend more toward it. In other words, the line created by the projected points is a little bit more 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/child gaussians with the function `Axis_SoftEnv_Ortho_Projection(`$\boldsymbol{p}_s^j$`)` that uses 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 influences 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 $\boldsymbol{\Sigma}_{sRef}$. The consequence is that it creates an offset in the natural gradient projection of $\boldsymbol{p}_s^j$ onto $\boldsymbol{p}_{sRef}$. In order to correct it, the same orthogonal projection function `Axis_SoftEnv_Ortho_Projection(`$\boldsymbol{p}_{sRef}$`)` is used on the reference sample $\boldsymbol{p}_{sRef}$ to calculate the "offset" $\boldsymbol{v}_{offset}$ which will be added to the center $\boldsymbol{\mu}_g^j$ before being 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 often chosen small to avoid divergence. In addition of being slow, the number of projected points can also be too dense, which might create an unwanted biasing of the tree growth toward some specific regions other than the child gaussian.
Since the goal $\boldsymbol{p}_{sRef}$ of the natural gradient projection is different at new each iteration, we use an adaptive step-size for each new projection by using the same formula illustrated in [21] 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 calculated poses at the beginning are more distant. In fact, if the first next pose updated by the natural gradient converges towards $\boldsymbol{p}_{sRef}$, then the rest of the projection will also converge for the same step-size because the Mahalanobis distance from each new updated pose to $\boldsymbol{p}_{sRef}$ with $\boldsymbol{\Sigma}_{sRef}$ is decreasing.
Since the context of application of this formula is different, we implement an additional step to avoid divergence of the projection. If the Mahalanobis distance using the first updated pose $\boldsymbol{p}_s^{j+1}$ to $\boldsymbol{p}_{sRef}$ with $\boldsymbol{\Sigma}_{sRef}$ is bigger than the Mahalanobis distance using 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 for N iterations 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 iterations, the criteria is still not met, then the projection is not evaluated (algorithm 3 line 19-21).
A comparison with a projection using a stepsize found with backtracking line search is pictured in figure 7

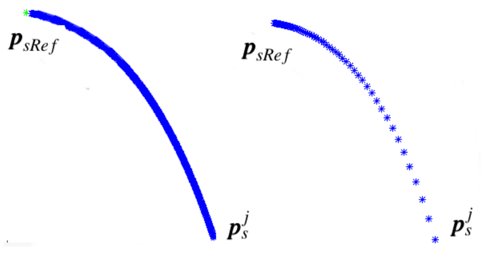

Fig. 7. Natural gradient projection of $\boldsymbol{p}_s^j$ onto $\boldsymbol{p}_{sRef}$ with: (left) stepsize found with backtracking line search method, (right) our stepsize

III. Experiments and results

In this section, our objectives is to show firstly if our method can successfully be used to accomplish the task and secondly to uncover how our algorithm influences, with and without dimension reduction, the generation of an end-to-end trajectory w.r.t the intention of the user. As we assimilated his intention when demonstrating the skill to the biggest variance of the Gaussians in the model, we are interested to see how the end-effector moves accordingly and w.r.t the Soft-Envelope in order to avoid an obstacle. For this purpose, we tested our method with two different tasks with different models. Using these two tasks, the performance of our method is assessed and its runtime is also compared with the RRT-JT introduced by Vandeweghe *et al.* in [25], a planner renowned to be fast in the sampling-based planning literature. It uses the Jacobian transform to bias the growth of the tree toward a pose while exploring the C-space. For our experiment, we have made 3 modifications to adapt our problem in Lfd: the goal sample comes from the sampling inside the child Gaussian, the search for the nearest node when it is biased towards the goal sample and the criteria to end the planning are changed according to our method explained in section II-B.1. Otherwise the algorithm is the same as described in his paper (and thus does not used any constrained area like the Soft-Envelope).

For the experiment, we first used 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. 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.
We tested our algorithm with 2 simple tasks whose skill models, learned with 7 successful demonstrations, are only encoded with two Gaussians. 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. Moreover, the initial configuration of the arm is placed above the table with its end-effector inside the red Gaussian.

A. Task 1: Pick and drop

The task is to have the end-effector, that is holding a small object(cherry fruit), goes above a wine glass in front of it and release its gripper to drop the object inside. During the execution, the Cheez-It box is used to obstruct the arm movement as shown in figure 8.
The skill of the model is represented with the Gaussians in figure 9. We intentionally demonstrated it such that the robot has more freedom/variance along the long side of the

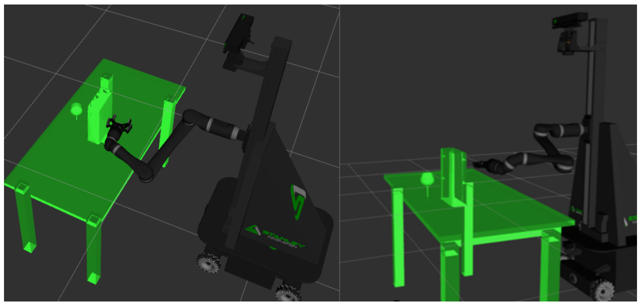
Fig. 8. Setup of the task 1: a plastic wine glass and a Cheez-it box on the table

table at its initial configuration (as shown with the variance of the red Gaussian). In the logic that the end-effector can release the object it is holding in a relative distance above the glass and still accomplish the task (as long as the object end up inside the glass), we show more variance in the vertical direction w.r.t to the glass/table (blue gaussian). In this task, our objective is to show how our algorithm with dimension reduction influences the trajectory generation toward directions reflected by the variances of the Gaussians in the model, even though their variance are already clearly defined . Results of the generated trajectories are pictured in figure 9

We can clearly see that all the trajectories with dimension

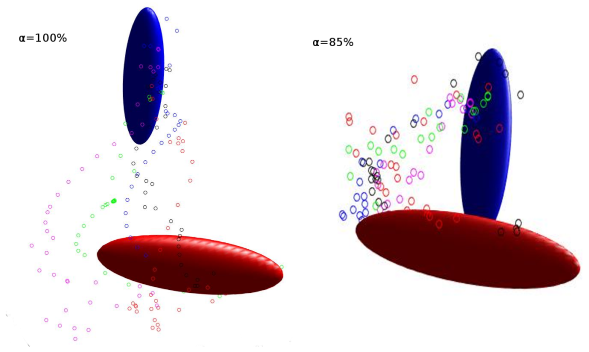

Fig. 9. Plot of 5 different trajectories (distinction by color) in x,y,z space for task 1 with (right image) $\alpha = 85\%$ and (left image)$\alpha = 100\%$

reduction first follow the variance of the first gaussian (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

The setup in task 2 is pictured in figure 10 in which we used the coffee can in the middle to hinder the motion. The task is to have the end-effector pick an object on the table

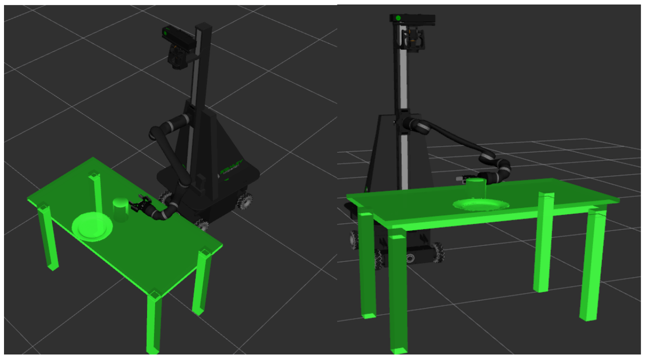

Fig. 10. Setup of task 2: a plate and a coffee can on the table

(a fruit for instance) and place it on a plate in front of it. For the demonstration, we wanted the end-effector to always begin at the same position above the table but with little more variance in the vertical direction w.r.t the table (red Gaussians). Then, when proceeding to deposit the object on the plate, it can cover it entire space and also release the object at certain height, which is why the blue Gaussians is spherical/ellipsoidal.
The objective in this task is to show how, with less obvious variance from the Gaussians (unlike the task 1), our algorithm encapsulates and uses their principal direction to influence the creation of trajectories.
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.

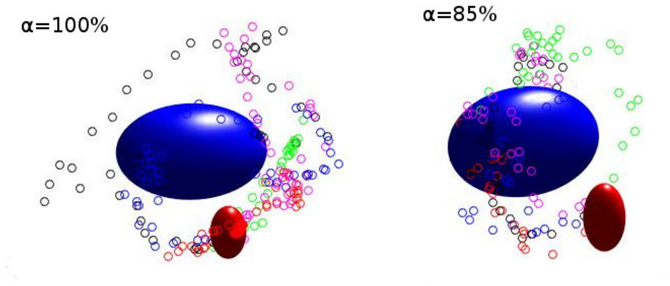

Fig. 11. Plot of 5 different trajectories (distinction by color) in x,y,z space for task 2 with (right image) $\alpha = 85\%$ and (left image)$\alpha = 100\%$

C. Tasks Performance

All the trajectory displayed in the previous subsections were able to successfully perform 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 12 for the first and second task.

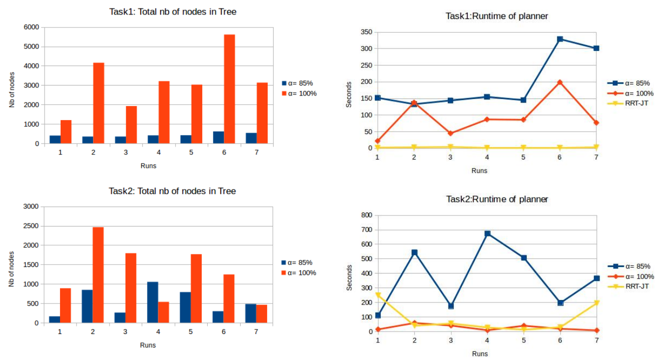

Fig. 12. Plots: total number of node in tree (col. 1) and Runtime(col. 2) for the first (row 1) and second (row 2) task

As expected, both tasks display similar performance results: it takes less time without dimension reduction because more joints configuration node, whose poses are inside the full Soft-Envelope, are used to grow the tree than with the reduced Soft-Envelope. Indeed, since the Gaussians of the Soft-Envelope are reduced and thus their size is smaller, the poses from the natural gradient projection belongs more easily to the full Soft-Envelope than to the reduced one, which is why the total number of nodes in the tree is in general bigger than the one with reduce dimension.

In addition, we also compare the speed performance of our method with the RRT-JT planner, known to be fast thanks to its biasing in workspace using the Jacobian transform. We can clearly see that it outperforms our algorithm with and without dimension reduction for the first task but holds similar performance or is even outperformed by our method without dimension reduction in the second task.
The reason for such fast planning for the first task is because of the variance of the child Gaussian (blue one) which is large in the vertical direction. Hence, using the Jacobian biasing method with the (goal) samples from this Gaussian is effective because it does not run into the Cheez-it box obstacle. In the case of our algorithm, it is slower because of all the projections for each sample inside the Soft-Envelope: even if more nodes are used to grow our tree for one sample inside the Soft-Envelope (used of intermediate projected joints angles whose poses are inside the Soft-Envelope to also grow the tree), they may not all lead the tree to converge towards the child Gaussian.
For the second task, RRT-JT is, for some runs, outperformed by our method because even if the goal samples from the child (blue) Gaussian cover some large space above the table, it is sometimes not enough. In fact, the biasing of the tree to these goal samples with the Jacobian transform makes the tree go straight into the obstacle. It is however still able to complete the goal thanks to its alternation from its biasing feature to a simple, complete RRT. In addition, there were few trials that we did not illustrated in the result graph because even after waiting for more than 5-6 minutes, the RRT-JT planner was still not able to find any solutions.

Even though this could also have occur with the first task, the difference is, in addition of the difference of child Gaussian variance, the position of the obstacle that is closer to the end-effector than the task 1, making the tree runs more into the obstacle when biasing it the Jacobian transform. With our method, the planner is able to quickly find a path thanks to the Soft-Envelope which indicates a way to quickly bypass the too close obstacle.

IV. Discussion

In general, our planner is not design to be fast, but depending on the scenario it can hold good performance as in task 2. However it introduces a rather a new approach, to the best of the author's knowledge, that is easily adapted to Learning from demonstration work with the skill encoded with multivariate Gaussians, to constrain an area in workspace, focus the search around it and toward one direction related to user intention and plan accordingly. Moreover, if speed is a concern, our method can easily be combine with Jacobian transform from the RRT-JT to benefit from its biasing features.
Although we have mentioned

V. Future Work

In this work, we have presented an approach that depends on the skill model to generate trajectory with sampling-based planner, and can further influence the planning by indicating one direction in 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 improve the algorithm such that it could plan a trajectory with dynamic obstacles (one direction of work is to use the RRTx suggested by Otte *et al.* in [22]).

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*, PCT-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 in simulation, showing the possible influences of the model on the motion of the end-effector.

VII. Acknowledgements

The author 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] F. Hoffmann T. Bertram A. S. Phung, J. Malzahn. Get out of the way - obstacle avoidance and learning by demonstration for manipulation.

[2] 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.

[3] 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.

[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. 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.

[6] 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.

[7] 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.

[8] S. Calinon. A tutorial on task-parameterized movement learning and retrieval. *Intelligent Service Robotics*, 2015.

[9] S. Calinon, F. Guenter, and A. Billard. On learning, representing and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man and Cybernetics, Part B. Special issue on robot learning by observation, demonstration and imitation*, 37(2):286–298, 2007.

[10] 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.

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

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

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

[14] 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.

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

[16] Florent Guenter and Aude G. Billard. Using reinforcement learning to adapt an imitation task.

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

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

[19] Sertac Karaman, Matthew R. Walter, Alejandro Perez, Emilio Frazzoli, and Seth J. Teller. Anytime motion planning using the rrt. In *ICRA*, 2011.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

[25] J Michael Vandeweghe, David Ferguson , and Siddhartha Srinivasa. Randomized path planning for redundant manipulators without inverse kinematics. In *IEEE-RAS International Conference on Humanoid Robots*, November 2007.