Motion Planning and Reactive Control on Learnt Skill Manifolds

Ioannis Havoutis and Subramanian Ramamoorthy

***Abstract*—We address the problem of encoding and executing skills, i.e., motion tasks involving a combination of specifications regarding constraints and variability. We take an approach that is model-free in the sense that we do not assume an explicit and complete analytical specification of the task - which can be hard to obtain for many realistic robot systems. Instead, we learn an encoding of the skill from observations of an initial set of sample trajectories. This is achieved by encoding trajectories in a *skill manifold* which is learnt from data and generalizes in the sense that all trajectories on the manifold satisfy the constraints and allowable variability in the demonstrated samples[1]. In new instances of the trajectory generation problem, we restrict attention to geodesic trajectories on the learnt skill manifold, making computation more tractable. This procedure is also extended to accommodate dynamic obstacles and constraints, and to dynamically react against unexpected perturbations, enabling a form of model-free feedback control with respect to an incompletely modelled skill. We present experiments to validate this framework using various robotic systems -ranging from a 3-link arm to a small humanoid robot- demonstrating significant computational improvements without loss of accuracy. Finally, we present a comparative evaluation of our framework against a state-of-the-art imitation learning method.**

I. Introduction

REALISTIC humanoid robotic tasks are often defined by skill specifications that involve a combination of high-dimensional nonlinear dynamics, kino-dynamic constraints and task variability that is often idiosyncratic when captured in the language of cost functions in an optimal control setting. Although, in principle, such robotic skills may be handled by well understood analytical methods, it can often be difficult and expensive to formulate models that enable such an analytical approach.

As an instance of a task that motivates our general approach, consider the Nao humanoid robot as used in robotic soccer competitions. The agent as a whole needs a variety of different skills involving locomotion in the presence of dynamic obstacles, full body manipulation such as for kicks and dives, etc. One specific instance of a flexible skill would be that of kicking - where the robot needs to be able to produce a variety of different kicking motions that cover the entire space reachable by the robot's leg, taking into account considerations ranging from staying upright, to dealing with constraints to leg and body movement, to the idiosyncracies of what defines a 'good' useful kick. The focus of this paper is on such skills that are variations to a basic type of motion, which we wish to encode in such a way as to enable efficient (in terms of computation time and performance) motion synthesis.

To recapitulate the standard approach to dealing with such problems, in a classical optimal control setting, e.g. Todorov and Li (2005), one typically begins with analytical models of the robot's dynamics and kinematics, as well as models of constraints such as due to joint limits or friction, the effects of impact, etc. In addition, one specifies the task in terms of objectives such as cost functions that specify in addition to time and energy considerations, preferences with respect to style, redundancy resolution and other somewhat more subjective factors. Armed with these, one is in a position to solve the problem using a variety of different search and optimization techniques. In practice, such an approach presents two types of difficulties. On the one hand, it can be hard to completely specify skills of the kind we have in mind in terms of the models and specifications mentioned above. However, even when this can be done, the problem of *searching through the possible trajectories* can be computationally intractable for many realistic problem instances.

In practice, motion synthesis problems are rendered tractable by the fact that the solutions to such optimization and search problems are not arbitrarily complex paths through high-dimensional configuration spaces but instead are typically restricted to subspaces that are defined by the task specifications. In a sense, all variations of the trajectories representing a parameterized skill can be described in terms of some qualitative structure that is common to the solution trajectories. Typically, this qualitative structure admits a geometric description and it can be learnt from data, which is the approach of this paper.

This observation regarding qualitative structure and its use in motion synthesis has a rich history. To identify a few key instances from this literature, Full and Koditschek (1999) demonstrate how biological motion is structured through collapse of dimensionality due to trimming away of degrees of freedom, exploiting synergies and symmetries. The resulting reduced model, a '*template*', can be used as the target for a reduced-complexity control strategy which may then be '*anchored*' in the original system. This has a parallel in robotics, e.g., Burridge et al. (1999), where complex control strategies are composed of simpler pieces that have well defined semantics in terms of stabilization within a basin of attraction. In Conner et al. (2009, 2003, 2006), we see an extension of this idea to the case of motion planning,

I. Havoutis is with the *Department of Advanced Robotics* at the Italian Institute of Technology, and S.Ramamoorthy is with the *Institute of Perception, Action and Behaviour*, School of Informatics, University of Edinburgh, Edinburgh EH8 9AB, United Kingdom. Email: Ioannis.Havoutis@iit.it, S.Ramamoorthy@ed.ac.uk.

[1]This paper unifies and extends previous work appearing in Havoutis and Ramamoorthy (2010a,b).

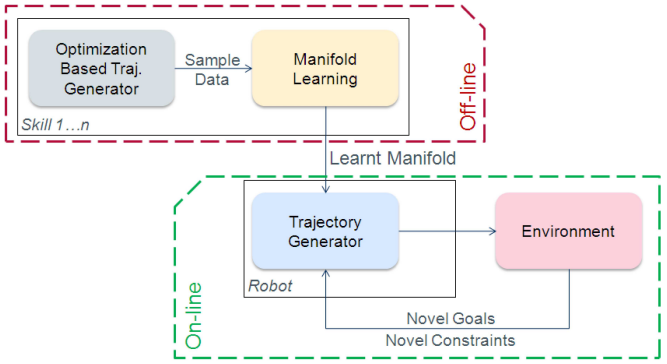


Fig. 1. Overview of our approach. Skills are encoded as skill manifolds that are learnt from example solution sets offline. The learnt skill set is the used by the robot to generate trajectories that accommodate novel goals and/or constraints, and to react to unforeseen disturbances in an on line manner.

where motion plans are composed from local parameterized feedback plans. Recent work such as Tedrake et al. (2010) brings together the continuous control and discrete motion planning angles by defining motion synthesis in terms of a tree structured object whose nodes are linear quadratic regulators with well defined domains of applicability.

A related observation from the control of autonomous flying robots, e.g., Dever et al. (2006, 2004) and Frazzoli et al. (2005, 2003), is that the solutions to optimal control problems are nicely ordered within maneuver sets. So, a lowest level description of a skill is in terms of a set of trajectories that capture the variability of the task while respecting all of the specifications regarding dynamics, constraints and costs. Such skills may then be composed and dealt with as discrete objects for higher levels of deliberation. For the purposes of this paper, we note that these sets typically admit a description in the form of manifolds embedded in the configuration and phase space of the system being controlled.

The focus of our work in this paper is on learning such solution sets from data in the form of a manifold, generalizing from a sample set of trajectories that share the underlying qualitative structure, and utilizing this manifold as the basis for subsequent motion synthesis, by a process of constrained geodesic trajectory generation. We learn the skill in an *offline* phase directly from demonstrated data. Such data can be either the result of an expensive optimization procedure (which is feasible offline) or demonstrations of an expert. The learnt skill manifold can be then used by the robot *online*, in an autonomous manner, accommodating novel constraints and goals, see Figure 1 for a visual depiction of this pipeline.

In our work, we bring together two different ways in which the concept of manifolds have often been used. On the one hand, control theorists and motion planning researchers are well aware that system dynamics of many interesting systems admit geometric structure yielding distinguished trajectories that are sub-manifolds in the configuration and phase spaces of the system. Indeed, some of our above mentioned examples are drawn from this literature, (e.g. Kobilarov and Marsden (2011), Lewis (2007)). Separately from this, machine learning researchers have looked at manifolds as a way of reparameterizing data, reducing from an initial high-dimensional description to some lower dimension that captures the modes of variation in a form that is suitable for visualization or operations such as clustering. Typically, in this way of using the notion of manifolds, one transforms the problem into the low-dimensional space and abandons the original high-dimensional space altogether as the objective was always the extraction of the low-dimensional coordinates and associated distances (e.g. Tenenbaum et al. (2000), Bitzer et al. (2008)). In this paper, we use the notion of a manifold in a sense that is closer to the control or motion planning usage - so we treat trajectories as being restricted to a subspace within a higher dimensional ambient space. Indeed, when we do reactive control in the face of disturbances, we use the learnt manifold to induce a distance in the ambient space. However, we also draw on the machine learning approach by utilizing those algorithms to identify the subspace without having recourse to an initial analytical characterization. So, a key aspect of the work described in this paper is the combination of these two notions within a novel framework for motion synthesis.

The remainder of this paper is organized as follows. The next section provides the background of our approach, covering both the motion planning and the machine learning domains. Section III introduces the manifold learning method that is core to our proposed framework. Section IV demonstrates how one can learn such a representation in a data driven manner, how to use a manifold encoding to generate novel trajectories, how to add novel constraints to such learnt skill manifolds, and how to utilize these for online reactive control. The next three sections (section V, section VI, section VII) present experimental evaluations on three robotic setups. The first is a planar 3-link arm suitable for demonstrating all the concepts of our approach in an easy to visualize manner. The second is a set of experiments on an anthropomorphic robotic arm. The third section evaluates our approach on a set of experimental setups using the Nao humanoid robot, performing tasks as walking and balancing on one leg. Finally, section (VIII) provides a comparison of our framework with a state-of-the-art *immitation learning* method, namely the *GMM/GMR* framework (Gribovskaya et al., 2010).

II. Background

At a high level, we are interested in solving the problem of finding optimal trajectories - where the specifications involve a variety of considerations already outlined above - in a way that mirrors the model-free nature of established algorithms such as for sampling-based motion planning. We would like to be able to minimize the artistry demanded of the user, by devising a reasonably general representation, an efficient learning method utilizing this representation and efficient planning methods that utilize this encoding. So, we begin by briefly discussing background in control and planning methods. Also, we are not the first to consider the notion of learning skills from demonstration, although we do present a novel take on the problem in terms of our task encoding and motion synthesis method. We review the prior work in this domain to set the scene for our own work.

Some of the most successful approaches to path planning are randomized algorithms that exploit sampling. For example,

the Rapidly Exploring Random Tree (LaValle, 2006) is a remarkably simple yet effective algorithm for planning a path between two points in configuration space. Many modern RRT-based samplers make use of two trees (Kuffner and LaValle, 2000, Diankov et al., 2008), rooted at the initial and the goal points, where in every step the trees are grown and a check for matching the trees is also performed. The key limitation, however, is that sampling a high dimensional space densely enough is computationally infeasible.

Other studies in this direction include obstacle based sampling (Thomas et al., 2007, Rodriguez et al., 2006), Gaussian sampling (Boor et al., 1999) and related approaches. Kuffner et al. (2002) demonstrated an interesting approach to this problem based on a combination of the RRT algorithm (LaValle and Kuffner, 2001) and post-processing in the form of dynamics filtering to ensure stability of the resulting behaviour.

Nonetheless, random sampling in high dimensions can be excessively wasteful when the underlying task has more restrictive structure. As it is becoming increasingly clear that many interesting robotic behaviours are restricted to low-dimensional subspaces (Vijayakumar et al., 2005, Bitzer et al., 2008, Full and Koditschek, 1999, Ramamoorthy and Kuipers, 2006, Jenkins and Mataric, 2004), for a variety of reasons ranging from joint limits and self-collisions to stability and energy constraints, it is desirable to leverage this in the process of achieving more focused coverage.

As known from the study of biological behaviours, natural systems utilize synergies and coordination strategies that allow for efficient locomotion and fast planning. Biological strategies usually have a musculoskeletal basis that is inherent to the dynamics of the system, that restricts movement to a subset of all possible solutions. In a robotics context, system and (possibly artificial) task constraints can serve the same purpose. Robotics (Ramamoorthy and Kuipers, 2008, Isto and Saha, 2006) and graphics (Safonova et al., 2004) researchers have utilized this fact to devise efficient motion synthesis strategies. Some recent works (Berenson et al., 2009, Berenson and Srinivasa, 2010, Stilman, 2007, Bretl et al., 2004) also address this issue by considering how task space constraints, e.g., end-effector constraints, can be used to structure planning in configuration space with local Jacobian mappings. However the low-dimensional nature of the solutions may not always be taken into account explicitly.

Along this direction, Kobilarov and Marsden (2011) considered optimal control of mechanical systems that evolve on a finite-dimensional Lie group. They use a discrete variational principle that yields a set of trajectories that approximately satisfy the dynamics and that respect the state-space structure (manifold) of the system in question, while they also developed an approximate numerical solution for the computation of such trajectories.

An aspect of the control problem associated with motion synthesis in complex systems such as humanoid robots is that of identifying desirable paths, executing evolution along these paths and correcting deviations away from these paths. In simple systems and systems where analytical models are available, there exist numerous control approaches that can provide principled and provably correct solutions (Stengel, 1995, Siciliano and Khatib, 2008, Levine, 1996). For many realistic robot systems, explicit design of controllers utilizing these established analytical methodologies appears infeasible.

Instead, one can break the state space of the system in smaller regions where a well defined control law can be implemented, this way assembling a global solution through local decompositions. For example, Zhang et al. (2009) perform an approximate cell decomposition of the free space and compute vector fields within cells by considering their adjacency relations, with the constraints and goals in mind. Conner (2008) decomposes the full space in smaller domains and sets up local *predefined* feedback laws with specific state-evolution properties. Such decomposition can also become difficult as the dimensionality of the space increases.

Imitation learning approaches attempt to encode motions, and feedback characteristics, directly from demonstration examples (Schaal et al., 2003). This sometimes requires little to no prior knowledge of the system characteristics while it is an intuitive way to teach complex motions with multi-DoF systems. Popular imitation learning paradigms include *Learning by Demonstration* (LbD) and *Dynamic Motion Primitives* (DMPs). Demonstration-Guided Motion Planning (DGMP) (Ye and Alterovitz, 2011) is another recent development on the imitation learning front.

LbD estimates the dynamics of a given set of demonstrations in a statistical manner, using Gaussian Mixture Model (GMM), approximating the behaviour with a nonlinear combination of Gaussian kernels. Gaussian Mixture Regression (GMR) is then used to sample the GMM and produce a control input given the current system state (Calinon and Billard, 2008, 2009, Hersch et al., 2008). Disadvantages of such methods include the appearance of spurious attractors that can trap the evolution of the state of the system and numerical difficulties that arise from matrix singularities, something we discuss further in section VIII.

DMPs provide a framework for learning the dynamics of demonstrated motions by using a simple dynamical system encoding with guaranteed attractor properties. Such dynamical systems though can only exhibit trivial behaviour, thus a non-linear function is being learnt from the examples that makes up for the idiosyncrasies of the demonstrated movements. The advantage of the formulation is that the generated trajectories retain the spatial and temporal characteristics of the motion that has been used for learning, providing discrete and rhythmic movement formulations, while allow for easy changes of start positions, goal positions and temporal scaling. In addition convergence to the goal is guaranteed by the dynamics of the underlying simple dynamical system, as the effect of the non-linearity vanishes with the evolution of the time variable (Ijspeert et al., 2002, 2001). A difficulty common to almost all imitation learning approaches is that of scaling up to systems with multiple DoFs and complex dynamics. Most recent results consider Cartesian space motions, limited to a 2 or 3 dimensional Cartesian space (Gribovskaya et al. (2010), Pastor et al. (2009)), making the variables in question much more "*well-behaved*", while an additional inverse kinematics layer is used to produce the actual joint movement of the plant.

The machine learning literature includes many examples

of dimensionality reduction methods used to abstract and/or make problem spaces manageable. For example Chalodhorn et al. (2006) use a low-dimensional sensory-motor mapping to optimize demonstrated motions over the robot's dynamics. In the same spirit (Abbeel and Ng, 2004) presented Apprenticeship Learning as an inverse reinforcement leaning formulation that approximates the unknown reward function of a *Markov decision process* (MDP) that the demonstrations are assumed to be optimizing. Wang et al. (2008) introduced GPDM, a Gaussian process based dimensionality reduction with a dynamical model of the evolution of the state, that can learn models of human kinematic trajectories. Bitzer et al. (2008) use a Gaussian Process-based nonlinear dimensionality reduction technique to arrive at an underlying model of demonstrated data, while using a parameterized path generation method over the learnt representation to generate novel movements. We contribute to this growing literature by explicitly addressing the issues of task variability and efficient feedback control despite incomplete model specifications.

A. Motivation and comparison to dim. reduction methods

One of our goals is to learn a geometric structure, i.e., a skill manifold, that captures the intrinsic structure of the space of trajectories by approximating the tangent space from demonstration data. So, if one begins with a set of motion examples from a specific class, e.g., due to a path optimization or redundancy resolution principle or even a more complex kinodynamic constraint, then one seeks a representation that intrinsically captures both the restriction of states to a low-dimensional space *and* the evolution of the trajectories in that space - as opposed to imposing a trajectory generation scheme, *post hoc*.

In the usual formulation, manifold learning is aimed at finding an embedding or 'unrolling' of a nonlinear manifold onto a lower dimensional space while preserving metric properties such as inter-point distances. Examples include MDS (Hastie et al., 2001), LLE (Roweis and Saul, 2000) and Isomap (Tenenbaum et al., 2000). Much of this work has been focused on summarization, visualization or analysis that explains some aspect of the observed data. On the other hand, as already mentioned, our goal is to learn a manifold encoding of motion tasks that allows us to perform control and planning operations in the spirit of above mentioned methodologies - which requires us to address a different problem from visualization or clustering.

Examples of state of the art manifold learning algorithms include the following:

- Coordinated factor analysis (CFA) has been introduced in Verbeek (2006) for modeling data sampled from manifolds. It uses a mixture of factor analyzers to approximate a non-linear factor manifold.
- Locally linear coordination (LLC) (Teh and Roweis (2003)) aligns the hidden representations used by each component of a mixture of dimensionality reducers into a single global representation of the data throughout space.
- Manifold charting (Brand (2003)) coordinates local parametric models to obtain a globally valid nonlinear embedding function. Like LLC, this charting method defines a quadratic cost function and finds the optimal coordination directly.
- Isomap, (Tenenbaum et al. (2000)), globally coordinates proximal pairwise distances using all-pairs shortest paths distances computed from a neighbourhood graph on the dataset. Isomap assumes underlying structure is manifested as a bordered manifold. This underlying manifold can be uncovered by Isomap, given the input data set is dense enough to cover the entire manifold and forms a single connected component.
- Locally smooth manifold learning (LSML) (Dollár et al. (2007)), rather than posing manifold learning as the problem of recovering an embedding, poses the problem in terms of learning a warping function for traversing the manifold. LSML explicitly focuses on generalizing to unseen portions of the manifold by making a smoothness assumption, something very beneficial for use in a robotics learning context as in our proposed framework.

Building on LSML, we learn a model of the tangent space of the low-dimensional nonlinear manifold, conditioned on the (temporally driven) adjacency relations of the high dimensional data. Such a learnt manifold model can then be used to compute geodesic distances, to find projections of points on the manifold and to directly generate geodesic *paths* between points. So, an important point that differentiates our approach from alternate data-driven approaches that also utilize some form of dimensionality reduction is that although we have a low-dimensional representation to reduce complexity, we solve an integrated planning and control problem in the ambient high dimensional space. This gives a clearer interpretation to what the controller is achieving: enforcing a large domain vector field *towards* the manifold and *along* the manifold. This makes the consideration of obstacles (Havoutis and Ramamoorthy (2010b)) and disturbances much more natural, without having to worry about how they themselves may be mapped to an artificial low dimensional space. Issues such as this latter point tend to be rather delicate in many alternate approaches.

III. Manifold learning

Our approach, building on LSML introduced in Dollár et al. (2006), attempts to learn an approximation of the tangent space that is locally common to neighboring data points. This way we can start from a point in the high dimensional data space and generate its neighbors, even beyond the support of the training data and up to a locally quadratic approximation. The key difference between embedding methods, that try to find a structure preserving embedding, and our approach is the goal of learning how to traverse such structure rather than embed it to a new coordinate frame.

A. Model formulation

Assume a given data set of D dimensions, in our case a set of example trajectories in a robot's state space. Due to task aspects including kinematic and/or dynamic constraints and task constraints, the data lies on a smooth d-dimensional manifold that is embedded in the D-dimensional state space.

The d dimensions of the manifold effectively explain the local modes of variation as presented in the dataset.

The data set can be described as a set of points $x \in \mathbb{R}^D$, while the image of the points on the manifold is $y \in \mathbb{R}^d$. There exists a continuous bijective mapping $\mathcal{M}$ that converts low dimensional points y from the manifold, to points x of the high dimensional space,

$$x = \mathcal{M}(y).$$

The central observation is that given two neighboring points on the manifold, x^i and x^j, the difference between these points, $\Delta^i_{.j}$, should be a linear combination of the tangent vectors at that point on the manifold, scaled by an unknown alignment factor ϵ^{ij}. Assume a mapping $\mathcal{H}$ from a point on the manifold to its tangent basis $\mathcal{H}(x)$,

$$\mathcal{H}: \; x \in \mathbb{R}^D \mapsto \left[\frac{\partial}{\partial y_1}\mathcal{M}(y) \cdots \frac{\partial}{\partial y_d}\mathcal{M}(y)\right] \in \mathbb{R}^{D\times d},$$

where each column of $\mathcal{H}(x)$ is a basis vector of the tangent space of the manifold at y, i.e. the partial derivative of $\mathcal{M}$ with respect to y. Taking $\Delta^i_{.j}$ to be the centered estimate of the directional derivative at $\bar{x}^{ij}$ (where $\bar{x}^{ij} \equiv (x^i + x^j)/2$) and ϵ^{ij} to be the unknown alignment factor, we have $\mathcal{H}(\bar{x}^{ij})\epsilon^{ij} \approx \Delta^i_{.j}$, that holds given ϵ is small enough and the manifold can be locally approximated with a quadratic form.

B. Formulation of the learning problem

We learn a model of $\mathcal{H}$ from data, parametrized by θ, giving us the tangent space model $\mathcal{H}_\theta$. Learning requires the following error to be minimized:

$$err(\theta) \quad = \quad \min_{\{\epsilon^{ij}\}} \sum_{i,j\in N^i} \left\|\mathcal{H}_\theta(\bar{x}^{ij})\epsilon^{ij} - \Delta^i_{.j}\right\|_2^2. \qquad (1)$$

The goal of training is to find the θ that minimizes Equation 1, where ϵ^{ij} are additional free parameters that are optimized over and do not affect model complexity.

To enforce the smoothness of the mapping $\mathcal{H}_\theta$ we add an explicit regularization term, in addition to implicit smoothness that may come from the form of the model itself. The intuition behind the first term is that the learned tangents at two neighboring locations, $\bar{x}^{ij}$ and $\bar{x}^{ij'}$, should be similar, i.e., $\left\|\mathcal{H}_\theta(\bar{x}^{ij}) - \mathcal{H}_\theta(\bar{x}^{ij'})\right\|_F^2$, should be small. The second term is used to avoid numerical instabilities, ensuring that $\mathcal{H}_\theta$'s do not get very small and ϵ's very large, thus $\left\|\epsilon^{ij}\right\|_2^2$ is also constrained. This way the following regularization term is added to Equation 1:

$$r \quad = \quad \lambda_E \sum \left\|\epsilon^{ij}\right\|_2^2 + \lambda_\theta \sum \left\|\mathcal{H}_\theta(\bar{x}^{ij}) - \mathcal{H}_\theta(\bar{x}^{ij'})\right\|_F^2 \qquad (2)$$

To simplify the error term we can rewrite it using a single λ as we can treat $\mathcal{H}_\theta$ and $\alpha\mathcal{H}_\theta$ as the same for any $\alpha > 0$. This way the error of $\mathcal{H}_\theta$ with regularization terms $(\lambda_E, \lambda_\theta)$, is equal to the error of $\alpha\mathcal{H}_\theta$ with regularization terms $(\alpha^2\lambda_E, \frac{1}{\alpha^2}\lambda_\theta)$, in essence translating in a single $\lambda_E = \lambda_\theta = \lambda$. Throughout this work we set λ equal to 10^{-3}.

C. Learning algorithm

In principle any regression technique can be employed for modeling $\mathcal{H}_\theta$, e.g. Vijayakumar et al. (2005), Rasmussen and Williams (2006). A linear model of radial basis functions (RBFs) is particularly well suited for the task (Bishop (2007)). The RBFs we use are of the form:

$$f^k(x) = exp(\frac{-||x - \mu_k||_2^2}{2\sigma_k^2}),$$

where the basis centers μ are computed with K-means clustering on the given dataset and the basis width σ is set to be twice the average of the minimum distance between each cluster and its nearest neighbor center. This is a common choice for σ in RBF methods (Bishop, 2007), that ensures coverage of the space between kernels, i.e. no disconnected components exist. From the RBFs we can compute a feature vector $\mathbf{f}^i$ from $\bar{x}^{ij}$ of the form $\mathbf{f}^i = [f^1(\bar{x}^{ij}), \ldots, f^k(\bar{x}^{ij})]$, where the number of basis function k controls the smoothness of the final mapping $\mathcal{H}_\theta$. This way a large k would result in a mapping that better fits local variations of the dataset but whose generalization abilities to other points on the manifold would be weaker, following the bias-variance trade-off. We can then define,

$$\mathcal{H}_\theta(\bar{x}^{ij}) = [\Theta^1\mathbf{f}^{ij}, \ldots, \Theta^D\mathbf{f}^{ij}],$$

where the Θs are randomly initialized $d \times f$ matrices.

To solve the system we compute the thin singular value decomposition (SVD) of Δ^i. From the linearity assumption there are at most d non-zero singular values. In practice we can compute the truncated SVD that keeps at most $2d$ singular values and allows for significant computational reduction. This way we have $\Delta^i = U^i\Sigma^i{V^i}^T$ and by plugging in this and $\mathcal{H}_\theta(\bar{x}^{ij})$ into Equation 1, and rearranging we get:

$$err(\theta) \quad = \quad \min_{E^i} \sum_{i=1}^{n}\sum_{k=1}^{D} \left\|{\mathbf{f}^i}^T{\Theta^k}^T E^i - U^i_{k.}\Sigma^i\right\|_2^2. \qquad (3)$$

To solve Equation 3 simultaneously for E and Θ is complex but by keeping either one constant we can optimize iteratively the other variable, in an *EM*–like fashion, becoming a least squares problem. This way we solve for E^i keeping the Θ^ks fixed as:

$$E^i \quad = \quad arg\min_{E^i} \sum_{k=1}^{D} \left\|{\mathbf{f}^i}^T{\Theta^k}^T E^i - U^i_{k.}\Sigma^i\right\|_2^2 \qquad (4)$$

$$= \quad arg\min_{E^i} \left\|H^iE^i - U^i_{k.}\Sigma^i\right\|_F^2 \qquad (5)$$

$$= \quad {H^i}^+U^i\Sigma^i. \qquad (6)$$

In the next iteration we keep E^i fixed and optimize for Θ^ks by:

$$\Theta^k \quad = \quad arg\min_{\Theta^k} \sum_{i=1}^{n} \left\|{\mathbf{f}^i}^T{\Theta^k}^T E^i - U^i_{k.}\Sigma^i\right\|_2^2 \qquad (7)$$

$$= \quad arg\min_{\Theta^k} \left\|\left({E^i}^T \otimes {\mathbf{f}^i}^T\right) vec\left({\Theta^k}^T\right) - vec\left(U^i_{k.}\Sigma^i\right)\right\|_F^2 \qquad (8)$$

Since $vec\left(U^i_{k.}\Sigma^i\right) = \Sigma^i {U^i_{k.}}^T$ the least squares solution for Θ^k becomes

$$vec\left(\Theta^k\right) = \begin{bmatrix} {E^1}^T \otimes {\mathbf{f}^1}^T \\ \vdots \\ {E^n}^T \otimes {\mathbf{f}^n}^T \end{bmatrix}^+ \begin{bmatrix} {\Sigma^1 U^1_{k.}}^T \\ \vdots \\ {\Sigma^n U^n_{k.}}^T \end{bmatrix} \quad (9)$$

We continue iterating the error minimization procedure until the system converges. The iterative nature of the algorithm does not guarantee the convergence to a global minima of the error function, thus a number of random restarts are performed to avoid bad local minima. In all experiments we performed 3 random restarts and continue with the model that achieves the smallest error. In practice the error difference between the random restarts was never significant and, to our knowledge, there is no evidence that we suffered from serious local minima issues.

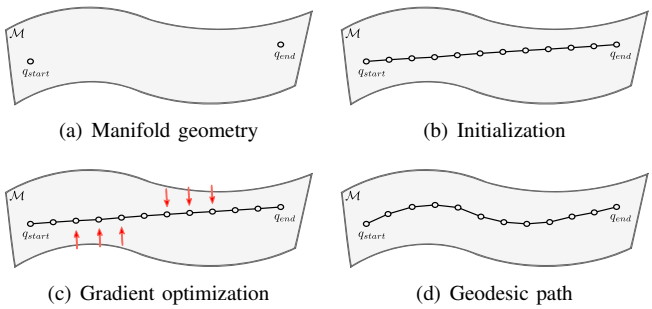
D. Intrinsic manifold dimensionality

The manifold dimensionality d is a parameter of the manifold learning approach. Throughout this work we have used a simple *cross-validation* approach to determine this parameter. We begin with a dimensionality of 1 and iteratively increase the model dimensionality while keeping track of the model error. The point at which the error derivative with respect to the dimensionality flattens is a good candidate for the model dimension. In other words beyond that point adding more dimensions to the model produces only a small decrement to the model error, sometimes also interpreted as *over-fitting*. A number of methods for estimating the intrinsic dimensionality of manifolds are available in the literature (e.g. Costa and Hero (2003), Kégl, Levina and Bickel (2004), Hein and Audibert (2005), Eriksson and Crovella (2012)). This is an active area of machine learning research, however, intrinsic dimensionality estimation is beyond the scope of this paper.

IV. Path planning and control with manifolds

The core of a planning and control framework requires a set of procedures relevant to learning, generalizing and executing motions. To begin with, a procedure for learning motions is important. In essence this would be a way to add new skill-representations to the existing skill-set. Another important feature of the framework would be to generalize from learnt skills and be able to refine them according to the constraints and goals of novel planning queries. In addition, the control phase of the framework requires a method for observing the nominal execution of the planned trajectories and reacting in a corrective manner should unexpected perturbations occur.

This section presents the main methods developed and utilized for learning manifold skill encodings and for planning and controlling a robotic system through such representations. The first subsection describes leaning skills from examples and generating unconstrained trajectories. The second subsection presents the method for incorporating novel constraints on learnt manifolds without the need to re-learn. The third subsection describes how we can control the system on-line given a learnt skill manifold. In essence we present step-by-step how a manifold representation can be used to tightly integrate path planning and control of robotic systems.



(a) Manifold geometry (b) Initialization

(c) Gradient optimization (d) Geodesic path

Fig. 2. Sketch of the geodesic trajectory generation procedure. (a) We begin with a learnt manifold model, a starting point on the manifold, often being the current state of the system, and a goal point, that is the state we want to reach. (b) We initialize with a trajectory that can traverse the ambient space of the system. This can be as crude as a simple manifold interpolation for low dimensional spaces. (c) We subsequently optimize the trajectory with respect to the manifold geometry by following the gradients of two errors defined over the geometry. The first drives the points on the manifold while the second minimizes the path length. (d) The outcome is an optimal length geodesic trajectory. In other words the shortest trajectory that connects the start and goal states and does not diverge from the underlying manifold geometry.

A. The manifold encoding for robotic skills

We are interested in preserving properties (e.g. spatial properties) of trajectories in the data set. Thus, our goal is to learn a model of the tangent space –a geometric representation– of the low-dimensional nonlinear manifold, conditioned on the adjacency relations of the high dimensional data. The learnt manifold can be used to compute geodesic distances, to find projections of points on the manifold and to directly generate geodesic paths between points.

1) Learning the manifold model: A manifold encoding is grounded in a set of example solutions. As mentioned earlier, such a set is produced either from a computationally intensive optimization procedure, an optimal controller, or even an expert demonstrator. The key characteristic is that such examples are trajectories corresponding to a specific objective that represents the skill, and these trajectories define the underlying manifold geometry. Thus, similar queries have similar solutions, directly deriving from a local smoothness assumption.

Training data are points from the system's state space that represent trajectories of a particular skill. These are of the form,

$$\mathbf{Q} = [\mathbf{q}^1, \ldots, \mathbf{q}^k]^T, \quad (10)$$

$$\mathbf{q}^i = q^i_1, \ldots, q^i_n, \; i = 1, \ldots, k, \quad (11)$$

where the number of examples is k and the length of each trajectory is represented by n. Each datapoint q^i_j belongs to a D-dimensional space, the system's state space, and lies on a locally smooth d-dimensional manifold. This subspace is embedded in the D-dimensional space. To approximate this we learn a mapping from each point of the D-dimensional space to the tangent basis of the manifold, $\mathcal{H}(\mathbf{q})$.

The datapoints belong to the state space of the system and such a state space can contain configuration variables, poses, higher order terms, as velocities and accelerations, or any combination of the above. Most of our experiments focus on the configuration space of the systems rather than the task space.

Modeling $\mathcal{H}_\theta$ is done with a linear model of radial basis functions (RBF's) with features over the evidence (Hastie et al. (2001)), where the RBF's, μ's and σ's are computed directly from the data in an unsupervised manner. The model is trained with the procedure presented in the previous section.

2) Optimal geodesic paths: Our goal is to find the shortest path between two prespecified poses $q_1,\ q_n \in \mathbb{R}^D$, D being the dimensionality of the configuration space, that respects the geometry of the learnt manifold. In a robotics context, being on the manifold essentially means that the constraints (e.g., optimality w.r.t. a particular task-specific cost) inherent in the training data are respected. In practice we, discretize our path into a set of n via points, $\mathbf{q} = q_1, \ldots, q_n$, with the q_1 and q_n being fixed, and we follow a combination of gradient descent steps to minimize the length of the path while not leaving the support of the manifold. Figure 2 presents sketches of each step of the trajectory generation procedure that is elaborated below.

The geodesic trajectory generation procedure consists of two phases. In the first phase an initial estimate of the path is created and in the second phase this path is optimized with respect to the manifold geometry and the overall path length.

3) Geodesic path initialization: The initialization procedure is based on the start and goal points, that are kept constant, and the average distance estimate ℓ, that is derived from the training data set. The process begins with the distance between the two initial points, the edges of the path. The distance is split in half and either the *left* or *right* edge (start or end) is grown recursively towards the middle.

Growing the path involves taking consecutive small geodesic steps, i.e. finding points that move towards the midpoint of the distance and are on the manifold. The point reached is set to the new estimate of *left* (or *right*) and the procedure continues with recursion. The recursion stops when the distance between the two points in consideration is equal or less to ℓ. Pseudocode of this phase is available in Algorithm 1 while Figure 2(b) provides a sketch of the outcome of the initialization step. This produces the geodesic path initialization which is subsequently optimized with respect to the manifold geometry. This recursion is guaranteed to find a path given the assumptions of the manifold learning phase described earlier (III-A,III-C), i.e. the manifold is a single connected component and smooth up to a locally quadratic approximation.

4) Geodesic path optimization: The initial estimate of the path is a manifold interpolation that roughly follows the manifold geometry but is not the shortest geodesic path possible. Since we have learnt the tangent space of the manifold we can find a minimum energy solution that is a geodesic path and its length can be minimized.

The optimization of the path is performed with an iterative gradient descent procedure that is performed in two steps. The first step follows the orthonormal (to the manifold) component

Algorithm 1 Initial Geodesic Trajectory

INPUT: $\mathcal{M},\ \ell,\ q_{start},\ q_{end}$
OUTPUT: $\mathbf{q} \equiv \{q_{start}, q_2, \ldots, q_i, q_{end}\}$
$qLeft = q_{start},\ qRight = q_{end},\ distPrev = \infty$
loop
  $diff = qLeft - qRight$
  $dist = norm(diff)$
  if $dist < \ell$ **then**
    $break$ {Distance reached}
  else if $dist < distPrev$ **then**
    $midpoint = (qLeft + qRight)/2$
    $qLeft = recursion(qLeft, midpoint)$
    $qRight = recursion(qRight, midpoint)$
    $break$ {Recursion}
  else
    $distPrev = dist$
    $qNext = qLeft\ (or\ qNext = qRight)$
    $H = \mathcal{H}_\theta(qNext)$
    $diff = H \times H' \times (diff/norm(diff))$
    $qNext = qNext + diff \times \ell$ {Geodesic step}
  end if
end loop

of the gradient of:

$$err_{\mathcal{M}}(\mathbf{q}) = \min_{\{\epsilon^{ij}\}} \sum_{i,j \in N^i} \left\| \mathcal{H}_\theta(\bar{q}^{ij})\epsilon^{ij} - (q^i - q^j) \right\|_2^2,$$

that essentially makes the q^i's "stick" to the learnt manifold by iteratively moving them to points where neighbouring (consecutive) bases are aligned. The second gradient descent step follows the parallel (to the manifold) component of:

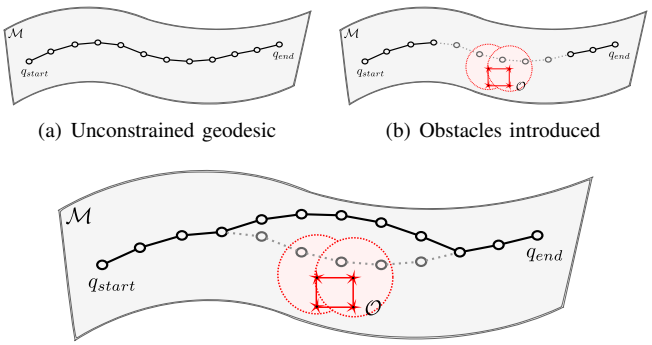
$$err_{length}(\mathbf{q}) = \sum_{i=2}^{n} \left\| q^i - q^{i-1} \right\|_2^2,$$

that iteratively minimizes the length of the path without leaving the support of the learnt manifold, while keeping the endpoints fixed.

Thus we are able to generate novel unconstrained trajectories that are geodesic paths over the learnt manifold. Such paths are locally optimal with respect to path length while also following the underlying geometry, specific to each robotic skill and observed in the demonstrated solution set.

B. Changing environments and dynamic constraints

Often in systems that act in a changing environment novel constraints can be introduced dynamically. For example consider the task of walking for a humanoid robot. We can learn an unconstrained walking manifold that produces stable stepping motions of variable step start and end points. Such trajectories assume that there are no obstacles in the way of the swinging leg. Now if obstacles are introduced, imagine the play-room of a kid, the system will have to generate motions that are able to avoid such obstacles in task space, e.g. step over a pile of Legos, avoid randomly dropped toys. Such obstacles were not present in the manifold learning step and are regarded as constraints that are not encoded in the skill manifold.



(a) Unconstrained geodesic (b) Obstacles introduced

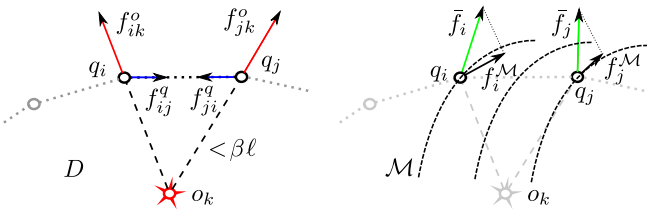
(c) Constrained geodesic path

Fig. 3. Example sketch of a constrained optimization scenario and of the proposed solution procedure. (*a*) An unconstrained generated geodesic trajectory for a path planning query originating q_{start} and reaching q_{end}. (*b*) The appearance of an obstacle set $\mathcal{O}$ introduces further constraints in the geodesic generation procedure as the intersection of the set with the manifold geometry, in effect, creates a patch that the solutions now have to avoid. (*c*) The obstacle set, $\mathcal{O}$, taken into consideration serves to drive the geodesic trajectory away from the red square obstacle. This is achieved through a *spring-system* model that extends the geodesic trajectory generation procedure and produces solutions that avoid such "*no-go*" while following the underlying manifold geometry.

Re-learning a representation that takes account such randomly occurring constraints would be infeasible in continually changing environments. Instead, our method reuses the previously learnt unconstrained manifold representations and incorporates new constraints in the motion planning phase. Figure 3 provides a sketch of such a scenario. In this setting we can generate unconstrained trajectories of the learnt skill manifold as explained previously (Figure 3(a)) but as the system's environment changes a set of obstacle points is introduced. What our method achieves is the generation of geodesic trajectories that can successfully navigate away from such an obstacle while not leaving the support of the learnt manifold, Figure 3(c). This is achieved with a spring-system model that optimizes geodesic trajectories with respect to novel constraints. The next subsection describes the procedure in detail.

1) Constrained geodesic paths: The algorithm presented in section IV-A2 generates unconstrained geodesic trajectories that generalize from the solution set presented to the manifold learning procedure in the training phase. In practice, we often require more control over the generated trajectories as, often, the system would need to avoid task space and joint space obstacles. This is a constrained trajectory generation problem over the manifold. We now describe a procedure for generating constrained geodesic paths that avoid *"no-go"* patches on the manifold surface. These are defined as sets of obstacle points $\mathcal{O} \in \mathbb{R}^D$ that are uniformly sampled from the *"no-go"* task space region and trace the obstacle patch in joint space. For example such points can be samples from the faces of a cube shaped obstacle or a set of points sampled from the surface of a sphere (Figure 3(b)).

The intersection of the manifold set and the obstacle set, $\mathcal{M} \cap \mathcal{O}$, is the region that we would like to take into consideration when generating a constrained geodesic path. This point set would drive the geodesic path away from the patch that



(a) Calculation of forces (b) Manifold projection

Fig. 4. (*a*) An obstacle point o_k affects only the path points that are within its range ($\beta\ell$) and exerts on them repulsive forces (red). In contrast the path points can exert repulsive (*not shown*) and attractive forces (*blue*) to their path neighbors. (*b*) All forces that act on each path point are averaged and the resulting mean vector is subsequently projected on the learnt manifold $\mathcal{M}$.

we want to avoid but given the learnt tangent space the path will not leave the surface of the manifold, thus properties that the manifold represents are still respected.

We treat the affected consecutive geodesic path points, $\mathbf{q}$, as a system of springs that can either exert attractive or repulsive forces to their neighbors. A force f^q_{ij}, with magnitude

$$\left|f^q_{ij}\right| = \ell - \sqrt{(q_j - q_i)^2},$$

between two consecutive path points q_i and q_j, is repulsive if the distance between them is less than ℓ, and attractive if the distance is greater than ℓ. The distance ℓ is a metric that is derived from the manifold learning step and is estimated directly from data (section IV-A2). The magnitude of the force is directly proportional to the distance of the points in question.

The obstacle point set, $\mathcal{O}$, exerts repulsive forces to the path points. The area of effect of the obstacle point set is also defined relative to ℓ; each obstacle point, o_k exerts a force f^o_{ik}, of magnitude

$$|f^o_{ik}| = \beta \times \ell - \sqrt{(q_i - q_o)^2},$$

to every path point, q_i, within a hypersphere of radius set to $\beta\ell$. The area of effect of $\mathcal{O}$ can be increased or decreased by tuning the scalar $\beta \in (1, +\infty)$ with obstacle clearance in mind, i.e. a small β will result in trajectories that evolve close to the obstacle set while a large β has the opposite effect.

We calculate all forces that act on each affected path point and compute a mean force vector for each point,

$$\bar{f}_i = \frac{1}{k}\sum^{k} f^o_{ik} + \frac{1}{j}\sum^{j} f^q_{ij}, \ (\forall j \in \{neigh(i)\}).$$

This vector is then projected onto the manifold, $f^{\mathcal{M}}_i = \bar{f}_i \mathcal{H}^q_\theta H^{q'}_\theta$, and each point is moved by a small step, γ, accordingly. In our formulation γ is statically defined in the interest of simplicity. Established numerical optimization methods (Nocedal and Wright, 2006) exist that can vary γ, thus achieving faster convergence, e.g a line-search step can be performed to compute the optimal step size for every iteration. Figure 4(b) provides a sketch of the average force projection step. In effect this makes the points take small geodesic steps away from the obstacle while not leaving the support of the manifold. We repeat the procedure until all path points have cleared obstacle points ($\mathcal{O} = \{\oslash\}$) or the algorithm has converged.

Algorithm 2 Constrained Geodesic Trajectory Generation

INPUT: $\mathcal{M},\ q_{start},\ q_{end},\ \mathcal{O}$
OUTPUT: $\mathbf{q} \equiv \{q_{start}, \dots, q_i, q_{end}\}$
$\mathbf{q} \leftarrow$ Optimal Geodesic Path($q_{start},\ q_{end}$)
repeat
  $d^{\mathcal{O}} \leftarrow$ Compute Distances($\mathbf{q},\ \mathcal{O}$)
  $[f^o_{ik},\ f^q_{ij}] \leftarrow$ Calculate Spring Forces($\mathbf{q},\ d^{\mathcal{O}}$)
  $\bar{f}_i \leftarrow f^o_{i.} + f^q_{i.}$
  $\bar{f}^{\mathcal{M}}_i \leftarrow \bar{f}_i \mathcal{H}^{\mathbf{q}}_\theta \mathcal{H}^{\mathbf{q}\prime}_\theta$
  $\mathbf{q} \leftarrow \mathbf{q} + \gamma \bar{f}^{\mathcal{M}}_i$
  $\mathcal{C}^i \leftarrow \partial \mathbf{q}^2 / \partial s^2$ {Curvature}
  $\bar{\mathcal{C}} \leftarrow 1/n \sum \mathcal{C}^i$
  $\bar{\mathcal{C}}^{\mathcal{M}}_i \leftarrow (\bar{\mathcal{C}} - \mathcal{C}^i) \mathcal{H}^{\mathbf{q}}_\theta \mathcal{H}^{\mathbf{q}\prime}_\theta$
  $\mathbf{q} \leftarrow \mathbf{q} + \gamma \bar{\mathcal{C}}^{\mathcal{M}}_i$
  $\delta \leftarrow \mathbf{q} - \mathbf{q}_{old}$
until $d^{\mathcal{O}} = \{\oslash\}$ *or* $\delta \leq 10^{-6}$

2) Curvature smoothing: The above procedure only acts on the geodesic path points that are in the area of effect (distance $< \beta\ell$) of obstacle points. This tends to lead to trajectories that are not smooth when only small portions of the paths are considered. To alleviate this, we introduce a step that considers the full set of path points and interplays with the constraint optimization.

We use the path curvature as a measure for smoothing the generated geodesic paths in a structured fashion. We calculate the curvature, $\mathcal{C}$, over the discrete geodesic points, $\mathbf{q} = q_1, \dots, q_n$ as

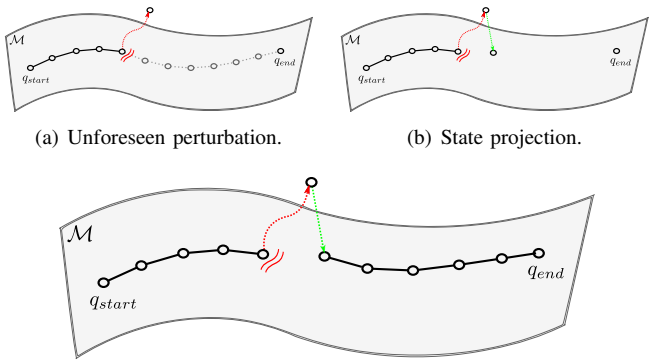
$$\mathcal{C}^i = \frac{\partial \mathbf{q}^2}{\partial s^2},\ i = 1, \dots, n-2,$$

where s is the distance between two consecutive points. We calculate the mean curvature $\bar{\mathcal{C}}$ and the error gradient $\bar{\mathcal{C}} - \mathcal{C}^i$ (vector), for each triple of path points. Each point is then moved by a small step along the error gradient and projected on the manifold tangent space. In general the curvature smoothing step is not mandatory but rather complementary to our obstacle avoidance formulation. The entire constrained geodesic trajectory generation procedure is summarized in Algorithm 2.

C. Control on and to a skill manifold

In the previous subsections we concentrated our efforts on solutions that restrain the system's state to evolve on the learnt skill manifold hypersurface. The intuition is that the set of solutions used as examples to our manifold learning phase, imply that the manifold geometry is indeed the set of desired states. Such a measure of "value" can be sometimes difficult to explicitly encode in terms of an analytical cost function, and in practice may encode task specifications, system constraints and general costs.

In effect, we demonstrated how a trajectory can be optimized to follow the manifold geometry from start to goal in a manner similar to a vector field on the learnt hypersurface. Perturbations that make the system's state jump to different points that belong to the manifold can be lazily handled with straightforward replanning, utilizing the tools presented earlier.



(a) Unforeseen perturbation. (b) State projection.

(c) On-manifold replanning.

Fig. 5. A sketch of an example where the ability to project would be necessary. (*a*) The system executes a geodesic trajectory when an unforeseen perturbation drives the state of the system to an off-manifold point. The remaining trajectory points are discarded. (*b*) Replanning from an off-manifold point would be insensible to the desired state evolution. Instead, we find the projection of the off-manifold state on the underlying geometry. This is the closest point that we then control for in a reactive manner. (*c*) A new geodesic trajectory is replanned, starting from the projection state and reaching to the goal state.

Imagine a scenario where a mobile manipulator would need to transfer a tray with drinks. The task manifold in such a case would include all these poses that keep the tray in a horizontal configuration and consequently the set of joint states that result in such an end-effector configuration. If the manipulator gets perturbed, e.g. a push from a guest who is not careful while the robot passes by, then the state of the system is likely to reach a point that lies beyond the support of the task manifold, i.e. the tray is tilted. In this case first our framework needs to recognize that the state has deviated from the nominal planned path and a reactive control input would serve to lead the state back to the desired manifold. The straightforward choice of action would serve to minimize the time that the state is in *off-manifold* space, thus would follow the shortest path to the manifold geometry.

It is often the case that perturbations will cause the state to leave the manifold (Figure 5(a)). Replanning from an off-manifold state would then be infeasible as the set of desired states is the manifold itself. A sensible choice is to find the closest on-manifold point and try to reach this in a reactive manner (Figure 5(b)). Once the state has returned to the manifold hypersurface, it becomes possible to replan as before (Figure 5(c)).

This empowers our framework with a global behaviour that covers the ambient space of the system in its entirety and enables us to reason quantitatively about the value (or cost) of off-manifold states. It can be viewed as endowing the space around the learnt hypersurface with a vector field controller that reactively seeks to return the state on the manifold should a perturbation occur. This is achieved via a projection operation as detailed in the following subsection.

1) Projection of states on manifold: We are given a learnt manifold model, $\mathcal{H}_\theta$, and a point q that belongs to the ambient space of the system. We seek a point q' that minimizes the distance between q and q', while q' must belong to the subspace that the manifold represents, i.e. q' is the closest

point on the manifold.

The projection of q on to the manifold $\mathcal{H}_\theta$ cannot be computed in closed form. Instead a gradient descent approach is utilized to find a new point q' on $\mathcal{H}$ that minimizes the distance, $d = \|q - q'\|_2^2$. First we need to initialize the procedure with an on-manifold point. One can use q' to be the nearest point in the training data or, in a control setting to the last state space point of a tracked trajectory, i.e. the point that the perturbation occurs and state of the system left the support of the manifold. Note that the distance formulation is not limited to the Euclidean norm as above. It can be substituted with any continuous and consistent distance metric.

Since $\mathcal{H}_\theta$ is defined over the whole $\mathbb{R}^D$ we calculate the orthonormalized tangent space at q', $H' \equiv orth(\mathcal{H}_\theta(q'))$, and $H'H'^T$ the corresponding projection matrix. We follow the gradient to the local minima on the manifold, using the update rule for q':

$$q' \leftarrow q' + \alpha H' H'^T (q - q'),$$

with α being a step size. The resulting q' is an on-manifold state that is closest, in a local sense, to the off-manifold state q.

D. Benefits of manifold control

Our primary objective in this setting is to devise a vector field in the configuration or joint space of the robot that enforces convergence to the skill manifold and approximately optimal evolution along the skill manifold. Beginning on the skill manifold, if the system were asked to achieve a goal that is infeasible (as indicated by the data), then one should be able to compute and execute a 'next-best' trajectory consisting of convergence to the desired subspace and subsequent evolution along it. So, the problem is essentially one of using the learnt structure to define an appropriately structured error function for control purposes.

Since all trajectories subject to the task specifications must lie on the manifold, the desired movement from off-manifold points is along the projection back to the manifold. Note that the precisely optimal corrective path segment may well be slightly different from this projection, depending on the specific nature of the overall task specification. However, this true underlying specification could not be known from data alone. So, the projection on to the manifold is the logical choice under this level of information. Deviations along the manifold, either due to dynamic obstacles or due to unforeseen perturbations, may be dealt with differently. Along the manifold, even if one were pushed away from the originally planned trajectory, one is still assured that the system is performing a reasonable movement subject to specifications. So, these

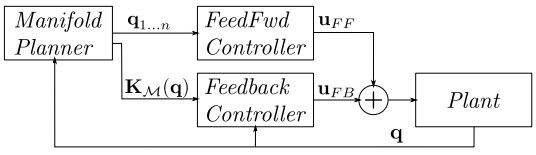
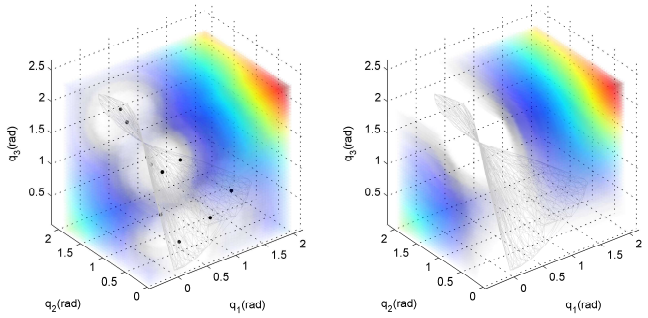


Fig. 6. A diagrammatic outline of the proposed control strategy utilizing the learnt manifold.



(a) RBF metric. (b) Manifold metric.

Fig. 7. *a*) Volumetric plot of the distance metric that can be evaluated directly from the model. The metric breaks down as the distance to the manifold geometry get smaller, leading to overestimation of the true distance. Black dots represent RBF centers, over which the evaluation is based. *b*) Volumetric plot of metric derived from our model. The metric evaluates the distance to the *modeled* surface and we see that it smoothly surrounds the underlying manifold. Distances range from dark blue (small) to red (large), while the closest distances are completely transparent for clarity.

two types of corrections may be handled differently, e.g. with different levels of stiffness.

Figure 6 outlines a controller architecture that achieves this type of behaviour. Geodesic paths along the manifold provide the feedforward component. Feedback corrects deviations along and away from the manifold - with different levels of gain. For sufficiently large perturbations away from the desired paths (such as in examples to follow), it may be more desirable to replan, in a receding horizon sense.

In addition, the feedback gain $K_\mathcal{M}(\mathbf{q})$ is state dependent and serves to scale the control input with respect to the systems distance from the *desired* manifold[2]. This coupling yields a system that is compliant with respect to *on-manifold* perturbations, that are in a sense indifferent to the task cost, but stiffens-up against perturbations that drive the state to *cost-expensive* parts of the space.

A key benefit of the manifold representation (as opposed to, say, a probabilistic model of possible velocities at each state) is that it provides a clear notion of deviation from skill sets and deviations within that set. With this, control is conceptually no more complicated than a simple proportional-derivative scheme but implemented in terms of a more sophisticated notion of 'error'.

In the absence of the manifold representation, one could still have implemented alternate error metrics such as, for instance, based on distances to centers of clusters of demonstrated data points. Stated in terms of our model, this is similar to defining errors in terms of the centers of RBFs used in our approximation. However, such a metric would yield suboptimal and non-smooth behaviours depending on parameters of the statistical model such as the number of clusters. We illustrate the behaviour of this metric within a bounded volume surrounding the desired subspace in Figure

[2] In the simplest case $K_\mathcal{M}(\mathbf{q})$ can be equal to the manifold metric. As with any feedback gain, this can be appropriately scaled for faster/slower system response.

7(a)[3]. What is plotted is the distance to the shortest kernel center, distances range from deep blue to red while the lower spectrum of blue is completely transparent for clarity. This results essentially in a number of low-distance spheres around the centers of the model. The metric becomes smoother as distances become larger but, on a local scale - which is the one of real interest for control purposes, such a metric would overestimate the distance to the manifold and be undesirably non-smooth. In contrast, by considering the distance to the *modeled subspace* directly we can get an accurate and smooth metric, that captures the distance accurately as shown in Figure 7(b).

An interesting possibility that arises from this (to be explored in future work) is that one could devise planning schemes based on variants of the A* algorithm, as the cost defined using the manifold is *admissible* - strictly less than or equal to the true cost of the underlying function. This property would not be present in the previous naive metric, which is prone to producing overestimates. In turn, the use of an admissible cost guarantees that A* (and variants) returns a path that is optimal and in practice greatly reduces running time.

In the following sections, we demonstrate the utility of these ideas for practical applications. The first example presents experiments on a simulated 3-link arm where both the manifold and the learnt model can be visualized. For the second example, we use the *Kuka LWR* anthropomorphic robot arm, with which we demonstrate how our method scales to more complex systems and more challenging tasks. Last, we present an example on the Nao humanoid robot, generating motions that stand stably on one leg.

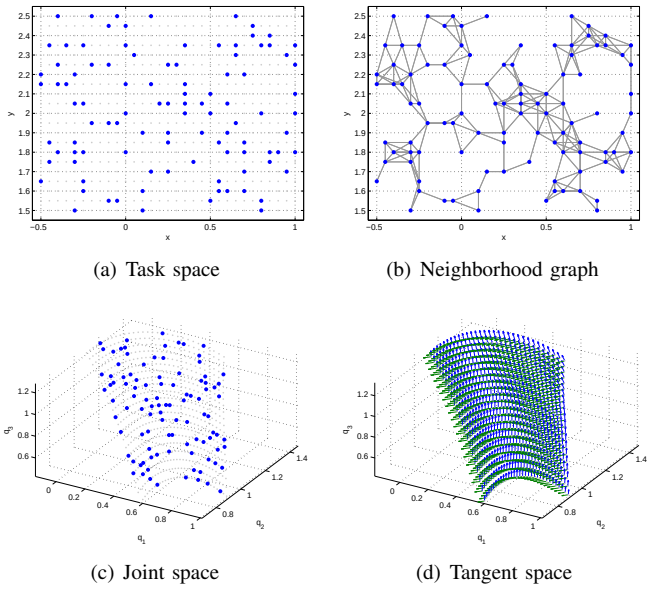
V. Experiments with a three link arm

A. Reaching with a robotic arm

Our first set of experiments are designed to elucidate the basic concepts underlying our approach. We use a *3-link planar* arm where we can explicitly visualize both the configuration space and the optimization manifold (surface that corresponds to a specific redundancy resolution strategy), along with possible obstacle points. The arm is a series of three rigid links, of $1/3$ length, that are coupled with hinge joints, producing a redundant system with 3 degrees of freedom (DoFs) that is constrained to move on a 2 dimensional plane (task space).

1) Reaching examples: We start with a 21×31 grid in task space and compute the joint positions for each goal point with an iterative optimization procedure detailed below. We subsample 100 grid points to get a random permutation for learning, as in Figure 8(a). We have chosen to start with training data that follow a grid in task space in order to visualize the corresponding geometry in the system's joint space. This structure arises from the optimization procedure that resolves the system's redundancy, i.e. a different redundancy resolution strategy would shape the state space geometry differently.

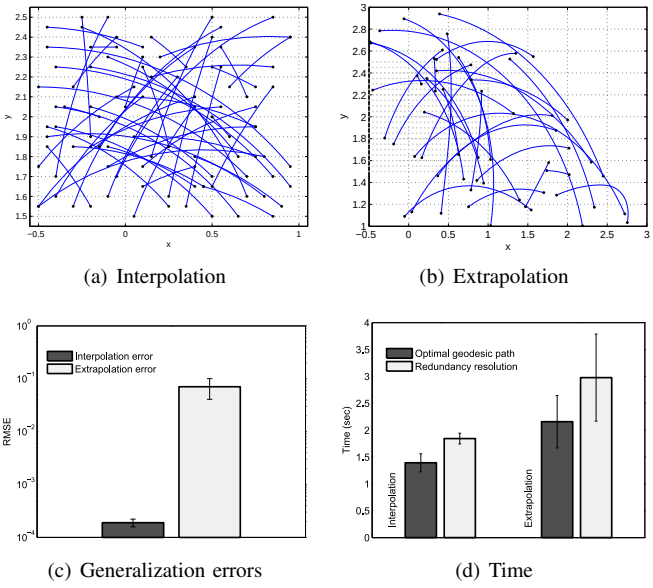
[3]The example here is based on our experiments with the 3-link arm. The gray trimesh represents the surface geometry generated by a minimum angles reaching behaviour. The details are clarified in the following sections.



(a) Task space (b) Neighborhood graph

(c) Joint space (d) Tangent space

Fig. 8. Learning the optimality manifold of a 3-link arm. (a) The planar task space of the arm and subsampled points (blue) used for leaning. (b) The neighborhood graph used for learning a manifold. (c) The optimality manifold that we wish to learn. Light gray points are not used for learning but are plotted to give a better estimate of the geometry of the manifold. Note that the manifold is not planar in the ambient space but twist and turns as we move down the q_3 axis. (d) The learnt tangent space model. Blue and green arrows are basis vectors evaluated at points that correspond to the original grid.



(a) Interpolation (b) Extrapolation

(c) Generalization errors (d) Time

Fig. 9. Results of the 3-link arm experiments. Novel task space trajectories produced with random start and end points where (a) demonstrates generalization within the region of support of the data, while (b) demonstrates generalization beyond the region of support of the training data. (c) RMSE error of generated trajectories against ground truth for the two cases. In the interpolation scenario the error is practically zero (y axis in log-scale). (d) Absolute planning time for the two cases. Note that in the interpolation case the length of the paths is consistently low.

In our experiment, we first have to choose a redundancy resolution strategy, which implicitly specifies the manifold that we will subsequently learn. Here, we choose the joint space configuration, $\mathbf{q}$, that minimizes the distance to a convenience (robot default or minimum strain) pose, $\mathbf{q}_c$. Formally,

$$\min \|\mathbf{q} - \mathbf{q_c}\|^2, \tag{12}$$

$$\textit{subject to } f(\mathbf{q}) - \mathbf{x} = 0, \tag{13}$$

where f is the forward kinematics and $\mathbf{x}$ is the goal endpoint position on the plane.

The resulting $\mathbf{q}$'s trace a smooth nonlinear manifold in joint space, depicted in Figure 8(c). We note that the manifold does not lie on a plane but on a convex strip that twists clockwise and tightens as we travel down the q_3 axis. Also different redundancy resolution strategies would produce different optimality manifolds. We note that, in general, this kind of information may not be explicitly known (in the case of human demonstration) or visualizable for more complex problems.

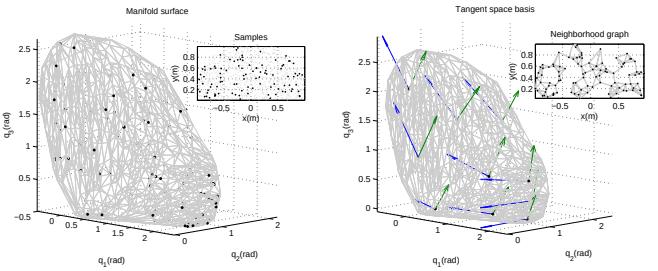
2) Implementation: The first step in data-driven learning of the desired manifold is to compute the neighborhood graph of the training data. We evaluate the task space distances to compute the neighborhood graph with the constraint that the graph contains a single connected component. In practice we gradually increase the neighborhood distance until all points are connected, as in Figure 8(b).

The tangent space that we wish to learn is inherently two dimensional. We learn a model of $\mathcal{H}_\theta$ with 10 RBF's and 100 points, the blue points in Figure 8(c). We can subsequently evaluate $\mathcal{H}_\theta$ at any point in our joint space. Figure 8(d) shows the tangent bases evaluated at every point of the previously generated grid. Note that the basis vectors are aligned and vary smoothly, i.e. we obtain a good generalization within the region of support of the data.

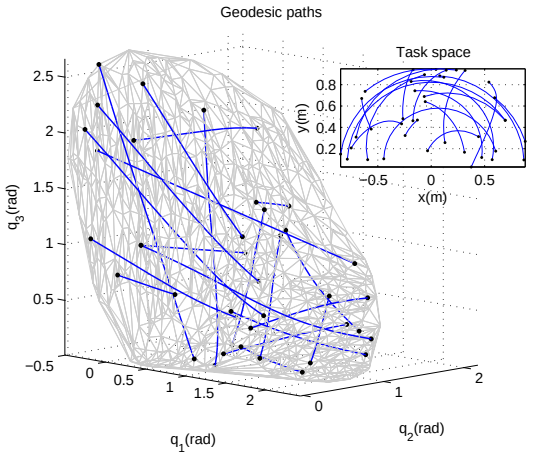
3) Generation of novel reaching solutions: For measuring the goodness of our learnt manifold, we use two metrics. Central to our aims is the generalization ability of the model. Thus we quantitatively evaluate the error of planned motions against the poses that the original optimization procedure would produce. We distinguish between two scenarios for our motion planning. The first evaluates the model's interpolation ability, generating trajectories that in task space lie within the grid from which 100 points have been sampled for learning. The second case evaluates the extrapolation ability of the model by generating trajectories, the endpoints of which lie outside the original grid. In both cases start and endpoint positions in task space were random, while results are averaged over 10 trials for each scenario.

We create 50 optimal geodesic paths, with random start and end points for each case, with the method detailed in section IV-A2. Samples of such paths for both generalization cases are depicted in Figure 9(a) and (b) (grid points in light gray for comparison).

We then collect all the intermediate points and compute the optimal solutions of their forward kinematics with the redundancy resolution algorithm detailed in section V-A1, as ground truth. We compute the *RMSE*, for each trial and for each case, between ground truth and prediction of model, for a total of 10 trials.



(a) Samples. (b) Learnt manifold.



(c) Generated geodesic paths.

Fig. 10. The manifold learning and usage for the 3link arm example. a) Starting with 100 datapoints in joint space, that correspond to task space coordinates as in the inset plot. b) The neighborhood graph in task space (inset plot), and the learnt tangent space that the model predicts for the RBF centers in the high dimensional space. c) Randomly sampled optimal (unconstrained) geodesic paths and corresponding task space trajectories in the inset plot. The thin gray trimesh is a densely sampled reconstruction of the underlying surface, used only for comparison and as a visualization aid.

The averaged errors are depicted in Figure 9(c). Note that the *RMSE* axis is in log-scale while the difference of the two bars is of 2 orders of magnitude. To be precise the average *RMSE* for paths generated within the region of support of the data is $1.8935 \times 10^{-4} \pm 3.6013 \times 10^{-5}$(practically zero), while beyond the support of the data the average *RMSE* is $6.84 \times 10^{-2} \pm 2.19 \times 10^{-2}$. In addition, computing the optimal geodesic paths takes less time on average (Figure 9(d)) in both cases.

B. Constrained reaching on a robotic arm

Following the preliminary evaluation of the previous section, we show that the method can handle a more interesting redundancy resolution strategy. To demonstrate this we begin with another reaching example where we also visualise the generated trajectories in the configuration space of the system.

1) Reaching examples: We randomly sample 100 Cartesian points from the top semicircle of the task space of the system. The dataset is 100 points of x and y pairs, where

$$\mathbf{x} = \begin{cases} x \in [-1, 1] \\ y \in [0, 1] \end{cases} \tag{14}$$

We run the task space dataset through an iterative optimization procedure detailed below and get the corresponding joint space datapoints, $\mathbf{q} = (q_1, q_2, q_3)$. A set of 100 such points is depicted in Figure 10(a)), as black dots in joint space and task space plots.

We densely sample the space with 900 more points that are used solely for visualization purposes and play no further part in the learning procedure. For visualization purposes, we use all 1000 points to compute a Delaunay triangulation of the joint space structure as sampled, and then plot the *trimesh* (triangle mesh) for comparison with the paths that our algorithm produces. This trimesh surface is depicted in all figures with thin gray edges.

We choose a different redundancy resolution strategy, implicitly specifying the geometry of the manifold (Figure 10(a), thin gray mesh). Here, we choose the joint space configuration, $\mathbf{q}$, that minimizes the absolute sum of joint angles, in a different view it minimizes the distance to a convenience (robot default or minimum strain) pose, $\mathbf{q}_c = (0, 0, 0)$, with an additional weighting on the cost of each joint movement, w_i. Formally,

$$\min \|w\mathbf{q} - \mathbf{q_c}\|^2, \tag{15}$$

$$\textit{subject to } f(\mathbf{q}) - \mathbf{x} = 0, \tag{16}$$

where w is a weighting vector, f is the forward kinematics and $\mathbf{x}$ is the goal endpoint position on the plane. We set $w = (4, 2, 1)$, which means that the cost of the first joint offset will be four times as significant as the last joints angle, thus penalizing more any motion of the first link. Such weighting is often used in robotics as, in a physical setting, moving the last link only is much more energy efficient than moving the first link, as the first link will have to move the rest of the system's weight as well.

The resulting $\mathbf{q}$'s trace a smooth nonlinear manifold in joint space, depicted in Figure 10(a). We note that the manifold surface resembles a convex strip that bends backwards towards the edges, much like a section cut of a bent tube. This is the surface that points of the specific optimality criterion trace. Also different redundancy resolution strategies would produce different optimality manifolds. We note that, in general, this kind of information is not explicitly known (in the case of human demonstration) or even visualizable, for many complex problems.

2) Implementation: As before we compute the neighborhood graph of the dataset and then learn the manifold model, $\mathcal{H}_\theta$ (Figure 10(a) and 10(b)). We can subsequently evaluate $\mathcal{H}_\theta$ at any point in our joint space. For example Figure 10(b) shows the tangent basis evaluated at the centers of the RBFs used. Note that the basis vectors are aligned and vary smoothly, i.e. we obtain good generalization within the region of support of the data. This way, in order to *"walk"* on the manifold we need to evaluate the learned tangent basis and follow each *local frame* for each consecutive step, in other words follow the blue and green arrows of Figure 10(b) for each point in question.

3) Generation of constrained reaching motions: Once we have learnt a model of the manifold tangent basis we have access to the geometric properties of the surface. Subsequently the geometry of the manifold can be used to generate optimal geodesic paths. The procedure for generating these unconstrained paths was described in section IV-A2, and the key advantage is that the generated paths will be of shortest distance and adhere to the manifold geometry. Optimal geodesic paths generated from randomly sampled start and end points are depicted in Figure 10(c), where the manifold geometry is also plotted (thin gray trimesh) for comparison. Note how the generated paths trace the underlying manifold geometry while also minimizing the deviation from a straight line connecting start and end points (non-geodesic minimum distance). The resulting task space trajectories –the geodesic paths run through forward kinematics– are also displayed in the inset plot at the same figure. Note that the resulting task space trajectories are curved, an observation discussed in the next subsection.

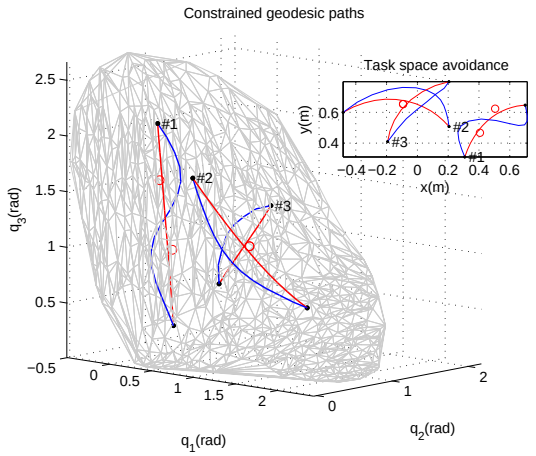


Fig. 11. Example geodesic trajectories that avoid point set obstacles on the learnt manifold. The unconstrained geodesic trajectories in red are the initial estimates that either collide or come unsuitably close to the obstacle points. The blue trajectories are the outcome of optimizing these geodesic trajectories with the constrained geodesic trajectory generation procedure that allows to avoid smoothly novel obstacle points in the system's state space. The inset plot demonstrates how the joint space trajectories are appear in the system's task space. See text for details.

In most realistic scenarios, we need more control over the geodesic paths that we generate. We would like to be able to specify patches on the manifolds that we would like the generated paths to avoid, while preserving their geodesic properties. This is accomplished by the procedure detailed in section IV-B1, and results are depicted in Figure 11. We start with a set of random start and end points and pick a list of obstacle points that intersect the manifold. In Figure 11 the points to be avoided appear as red circles, and effectively trace a patch that can be viewed as a *"no-go"* region on the manifold. The red lines are the predicted geodesic paths that travel through the obstacle regions. The blue lines are the constrained geodesic paths that are optimized with the obstacle patches in consideration. The resulting task space behaviour for this set of examples is visualized in the inset plot of the same picture.

4) Remarks: One observation, regarding the shape of the task space trajectories generated by geodesic paths, is that

the shortest path in the 3 dimensional joint space would be a straight line connecting the start and end points. The optimal geodesic paths are the joint space trajectories that connect start and end points and minimize the deviation from a straight line with respect to the manifold geometry. Now, the manifold is the surface defined as the union of all joint space paths that are optimal with respect to a specific redundancy resolution strategy. These are shortest paths that satisfy the optimality requirements implicitly encoded. In our scenario, the predicted trajectories would be composed of a series of points that minimize the weighted sum of joint angles, thus the task space trajectories would be subsequently optimized with respect to minimum angular change.

Another point is that the generation of geodesic paths is more efficient – and *much faster* ($\sim$ 1 sec vs. $\sim$ 3 sec) – than numerical optimization as described in section V-B1, while achieving (practically) equal results (approximation RMSE $\sim 10^{-3}$). In other words we are able to approximate the solution set of the costly optimization with an approximation that is able to generate accurate solutions in a fraction of the computational cost. In addition we are able to lazily add novel constraints and take them into account in the motion generative process without impacting the optimality with respect to the manifold hypersurface.

C. Manifold control on the 3-link arm

Firstly, we encode the task in a *skill manifold*, a subspace, in the underlying state space that is defined by the equivalence class of trajectories corresponding to various instances of the general task. Then, we define cost hypersurfaces that penalize deviations from this subspace of states within the ambient space. For instance, if a task is defined by the need to gyrate ones body in a certain style of configurations then all variations on that style are captured as trajectories along the skill manifold. Then, deviations from that style of gyration (i.e., away from the set of all feasible trajectories) would be penalized according to the specific structure of the task - by defining cost hypersurfaces with respect to that underlying skill manifold. This yields a vector field in the *ambient* space that constitutes both a basic plan from an initial condition and a controller to counteract perturbations, as also discussed in section II-A.

1) Implementation: We use the implementation that we presented previously in section V-B2.

2) Evaluation: To evaluate the accuracy of the model we randomly pick 100 start and end points and plan a trajectory between them, first with our method and second, with a naive quintic polynomial method as in Craig (1989) - our chosen benchmark. We distinguish two cases; an unperturbed trajectory, and a *random perturbation* occurring at $t = 0.25$ (Figure 13). We calculate the average cost per trajectory and average over the results for each case (Table I). The evaluation shows that with the use of the manifold controller we achieve consistently lower cost trajectories, while the difference is multiplied in the case where a perturbation occurs. The interpretation being that the naive planner is forced to stay in a high cost patch of the state space while the manifold finds the appropriate short path to the cost-optimal surface.

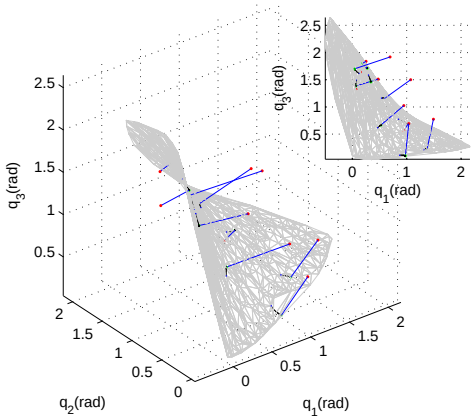
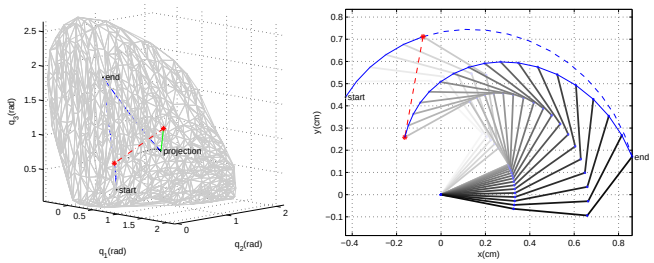


Fig. 12. Randomly sampled points in state space (red) and corresponding manifold projections (green). The inset plot is a different perspective of the same figure. Note that these projections are indicative of the control action, i.e., control vector field, which is naturally different in different regions of the ambient state space. (The thin gray trimesh is a densely sampled reconstruction of the underlying surface, the extra points being used only for comparison and as a visualization aid.) A rotating 3D version of the plot is also available in the accompanying video.



(a) Perturbed geodesic path. (b) Perturbation example projections.

Fig. 13. A typical trajectory resulting from this method. A geodesic path from start to end is computed, with a random perturbation occurring at time $t = 0.25$ that pushes the state away from the manifold. This new state is projected back on to the manifold to find the closest feasible state. A path from the projected point to the goal is then executed before continuing along. The task space trajectory with perturbation. The dashed blue line is the initial predicted trajectory while the red line is the motion due to the (severe) perturbation occurring at the first red star. The state is then pushed away from the initial trajectory and a new path to the goal is replanned after the novel state is projected on the learnt manifold.

VI. Experiments with the Kuka Lightweight Arm

A. Serving example with the Kuka Lightweight Robot arm

The next set of experiments demonstrates how our method can scale up to a higher dimensional problem and capture more interesting behaviours. For this, we use the 7-DOF *Kuka Lightweight Robot (LWR-III)*, shown in Figure 14(a).

The chosen movement corresponds to that of carrying a tray (it helps if we imagine that it may be loaded with high tumblers) from a randomized start position to a randomized goal position while trying to minimize total joint motion. This task can be broken down into two costs that need to be simultaneously optimized. The first of these must penalize deviation from a flat end-effector configuration, while the second must minimize the angular displacement.

This problem is defined as:

$$\min(J), \tag{17}$$
$$\textit{subject to } f(\mathbf{q}) - \mathbf{x} = 0, \tag{18}$$

where the cost, J, can be separated into two factors: $J = J_1 + J_2$, of the form:

$$J_1 = w\mathbf{q}^2, \tag{19}$$
$$J_2 = T^2_{pitch} + T^2_{roll}. \tag{20}$$

Where T is the end-effector orientation with respect to the global frame of reference. As the system is redundant we use a non-linear optimization method to obtain random training points.

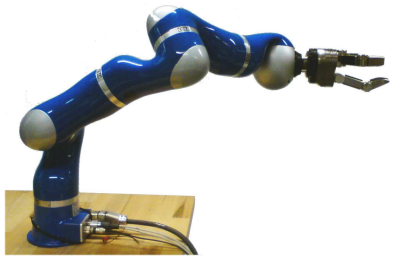
1) Implementation: As in the 3-link arm example, we start with the nearest neighbour (*NN*) graph computation, where we gradually increase the neighbourhood distance until no disconnected subsets exist. An *NN* graph is shown in Figure 14(c), where we plot the end-effector positions that correspond to the sampled configurations and the graph edges that result from the computation.

Our training set in this case consists of 100 data points that have been collected by sampling random end-effector positions and then optimizing with the procedure described earlier, Figure 14(b). The dimensionality of the system in this case is quite high, thus the sampling is necessarily sparse. The dimensionality of the manifold has been set to 4, while using 20 RBF kernels, as lower dimensional models did not achieve an acceptable test error.

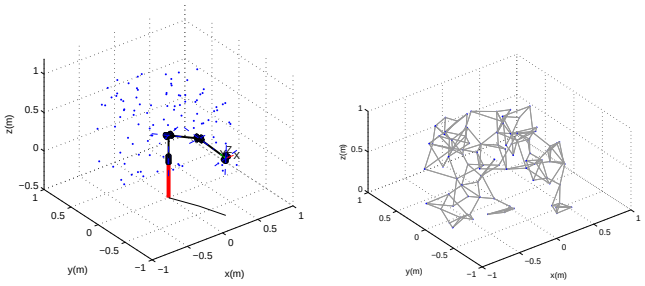
The dimensionality of the system prevents any meaningful visualization or qualitative observation about its geometry. Nonetheless the evaluation presented below reveals that the learnt manifold is an accurate model of the cost metric present in the demonstrated data, while planning and replanning with this model is highly beneficial.

2) Evaluation: The experimental procedure was as follows. We collect a set of 100 joint space trajectories that are produced by our method where the start and goal points are sampled randomly from the reachable space of the system. We further generate trajectories that correspond to the same start and goal positions using interpolation with quintic polynomials as in Craig (1989). We generate a random perturbation at time $t = 0.25$ and replan with both methods (Figure 15). We evaluate the true cost for all sets and compute the average cost per trajectory.

By comparing the resulting average costs we can see that our method produces significantly lower cost trajectories, i.e. the deviation from a flat end-effector configuration is lower while the angular displacement cost is also lower. The resulting benefit is magnified in the case where the trajectories are perturbed as the resulting average costs show (Table I), where the naive method on average tilts the end-effector (tray) by $0.55rad$, which would be considered a task failure. This occurs because in our method the system seeks to return to the space of the demonstration data as soon as the perturbation ceases and replans thereafter while following the manifold of (approximately) optimal cost. The naive method seeks to reach



(a) Robot.



(b) Samples. (c) Neighborhood graph.

Fig. 14. The redundant *Kuka Lightweight Arm*. (*a*) Picture of the robot. (*b*) The kinematic model of the Kuka LightWeight Arm along with 100 randomly sampled joint space endpoint positions. (*c*) The neighborhood graph that results from the sampled joint space points.

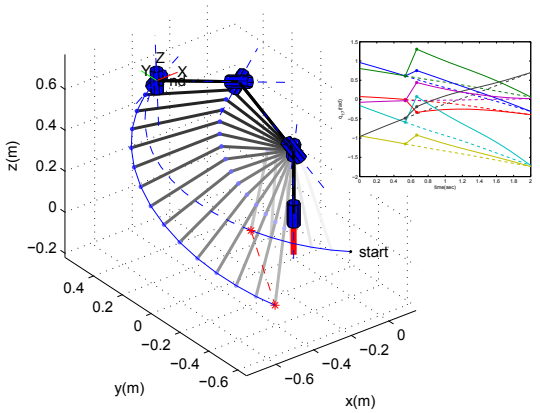


Fig. 15. An example planned trajectory. The unperturbed trajectory appears as a dashed line. The perturbation pushes the state away from the planned trajectory (red line). The solid blue line, originating at the point when the perturbation ceases (red star), is the replanned trajectory that extends from the projected state point and ends at the goal position. *Inset plot*: Example joint space trajectories including a large perturbation.

the goal without considering the underlying cost-optimal sub-structure thus is prone to spend the remaining time in a non-optimal part of the state space. In table I the evaluated cost for the predicted trajectories is also broken down to its two components, the average angular displacement (J_1) and the average displacement from a flat end-effector configuration (J_2). Both metrics assert that the manifold method provides superior results regarding the underlying cost metrics.

Figure 15 shows a typical run of the control strategy. Random start and end points are selected and we use our manifold-based method to generate a trajectory that reaches the goal while satisfying the learnt cost (dashed blue line). At

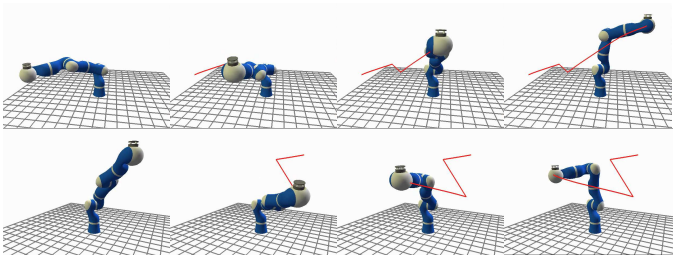


Fig. 16. Two examples of perturbed trajectories with the Kuka LWR arm. The red line traces the path of the end-effector (the perturbation is seen as the discontinuity in this trace). Time flows from left to right. Also available in the accompanying video.

time $t = 0.25$, a random perturbation occurs that pushes the state of the system away from the planned path. The controller recognizes the discrepancy between the planned and current state and computes the projection of the new state space onto the manifold, along a direction that is orthogonal to cost hypersurface. In other words, the system moves back towards the closest point that satisfies the learnt cost modeled by the manifold. A new path is replanned from this state towards the goal, the solid blue line originating from the red star and reaching the goal. Figure 16 contains snapshots from example trajectories on a realistic simulator[4]. See accompanying video for further examples.

The computational cost of the proposed procedure scales linearly with the size of the model, i.e. the complexity is coupled with the number of RBF kernels used. Planning a trajectory on average requires less than $0.1sec$ while projections from random state space points on the learnt manifold on average require $0.09 \pm 0.05sec$, on standard commodity hardware running not particularly optimized code. This suggests that manifold (re)planning can be a viable solution for online usage.

As a final remark regarding evaluation, we note that our manifold learning algorithm was provided with data from analytically optimized trajectories rather than human demonstration. We choose to do this with the aim of having precisely controlled experiments with clear ground truth. So, we are able to make concrete statements about the ability of our methods to recover the 'correct' solutions. As such there would be no difference if we replace these trajectories by, e.g. motion capture data, although we wouldn't be in a position to make similarly quantitative statements regarding performance.

VII. Experiments with the Nao humanoid robot

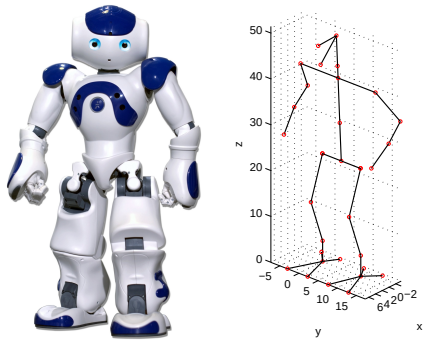
This section presents how our framework can be used with a humanoid robot. We show that we are able to learn, from a limited number of stepping examples, a representation that captures an approximately complete set of quasi-static walking motions. Furthermore, the produced motions can accommodate novel starting positions and goals while producing good (stable) generalizations of the examples provided.

[4]The simulator is developed in *SLMC* and uses a rigorous analytical model of the robot's dynamics. Note that the dynamics model is used for physically realistic simulation but is *not* available to the learning, planning or control algorithms.

TABLE I

Mean costs evaluated against the true cost functions. The results are mean±standard deviation over sets of 100 random sampled trials.

System	*Method*	*Unperturbed traj. cost*	*Perturbed traj. cost*
3-link	*naive*	0.9239 ± 0.1799	1.401 ± 0.3610
	manifold	0.8724 ± 0.1723	1.214 ± 0.2597
Kuka	*naive* (J_1)	0.7952 ± 0.0713	0.8173 ± 0.1215
	manifold (J_1)	0.6719 ± 0.0777	0.6862 ± 0.1317
	naive (J_2)	0.0154 ± 0.0082	0.5546 ± 0.1199
	manifold (J_2)	0.0119 ± 0.0072	0.0785 ± 0.0160



(a) Robot (b) Model

Fig. 17. The Nao humanoid robot used, (a) physical robot and (b) skeleton model.

Our experiments are based on the *Nao* (Figure 17) humanoid robot, popularly known as the chosen robot for the *Standard Platform League* in *RoboCup*. The *Nao* is approximately half a meter tall and weights $4.3kg$. It has an $500MHz$ *AMD Geode* processor onboard, running Linux. It has 25 DoFs and a variety of sensors. From the point of view of motion synthesis, it is an inherently unstable system, with an elevated center of mass.

A. Walking with NAO humanoid robot

We do not have an analytical model of the dynamics of the system. Even if we were to develop an approximate model, it would need to account for varying model parameters, e.g. change in the motor behaviour as the battery gets depleted or motor temperatures vary. Such effects are very hard to capture analytically, although they do matter in practice as we find from extensive experience within our laboratory. So, we prefer to work directly from experimental data. However, we do use a model of the robot kinematics for calculating the relevant foot and pelvic positions in global coordinates. It is worth noting that the kinematic model is used in a *black-box* manner and only for the evaluation of poses - it is not used by the learning, planning or control methods that we describe in this work.

We focus on the task of walking, with the aim of generating a motion synthesis strategy that achieves full coverage of a reasonably large interval in step length, width and height. In effect, the optimality surface would be the set of all solutions to all possible task space queries, thus a tangent space point would have a local coordinate frame that guides the path in that particular neighborhood. We begin with a redundancy

resolution strategy that would yield walking examples as training data for manifold learning.

1) Quasi-static walking examples: We frame the redundancy resolution strategy as a constrained nonlinear optimization problem. Algorithmically, we use a sum of squares (SoS) approach that uses the trust-region-reflective algorithm.

The optimization problem is of the form,

$$\min_{q} \mathcal{J}(\mathbf{q}), \tag{21}$$

$$\mathcal{J} = J^1(q) + \ldots + J^n(q), \tag{22}$$

$$\textit{subject to } f(\mathbf{q}) - \mathbf{x} = 0, \tag{23}$$

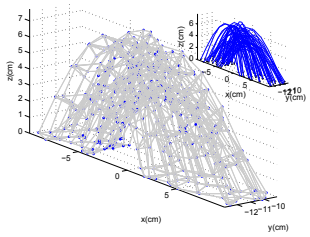
where $\mathcal{J}$ is the cost function that is composed of a number of cost factors J^n, f is the forward kinematics and $\mathbf{x}$ is goal task space positions. The cost function is a mixture of task constraints and stability constraints. The cost function we used for data generation evaluates:

- the task space distance between swing foot and sampled goal
- the alignment between swing foot and x/y versors, that keeps the swing foot flat with respect to the ground plane
- the deviation in pelvis position from the support polygon that the stance foot provides
- the alignment between waist and z versor, pushing the pose to an upright position.

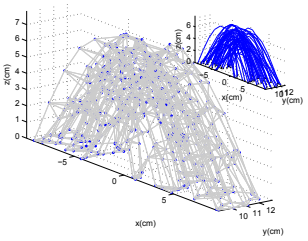
The initial pose for the numerical optimization algorithm is a default robot initialization pose with slightly bent knees.

To generate a walking trajectory we start with the desired task space path of the swing leg and the position of the pelvis, and discretize to 10 points. The swing foot trajectories are straight lines from start to goal points while the height of the foot is regulated with a sinusoid with varying apex height. In practice we set the position of the pelvis to be over the support foot and perform a double support weight shift step once the swing leg has reached the goal position. Lastly, we run the optimization procedure described earlier, and get the joint space trajectory of the leg swing and the weight swift phases for each complete task space step path.

The optimization results are approximately constant speed quasi-static trajectories, in the sense that inertial effects are negligible. We collected 50, full body, joint space trajectories for stepping with the right leg and the same amount for stepping with the left leg. Start and goal points of every step have been randomized within a reasonable reaching distance. The inset plots of Figure 18(b) and 18(a) show the task space trajectories of each swing leg foot midpoint, by running the datasets through the forward kinematics of the system.

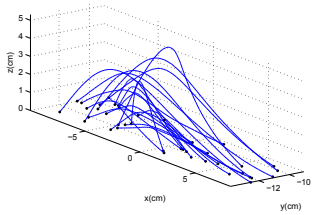


(a) Right foot training set

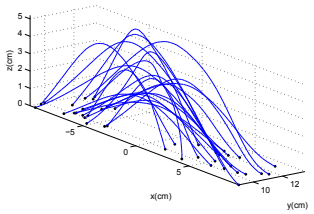


(b) Left foot training set

Fig. 18. The neighborhood graph, computed for a dataset of 500 points for each swing leg. Both plots show the swing foot midpoint position in task space. The inset plots show the corresponding continuous trajectories in task space.



(a) Generated right foot traj.



(b) Generated left foot traj.

Fig. 19. Generated (unconstrained) task space trajectories from randomly sampled start and end points. The trajectories correspond to swing foot midpoint trajectories in task space and are stable on the robot.

2) Implementation: Compared to our previous examples, this is a higher dimensional space and sampling is necessarily somewhat sparse. Of the 25 DoFs of the robot, we focus on the 12 DoFs for legs and hips, keeping the arm and head joints at a constant pose. Furthermore we separate each footstep into a swing phase and a weight shift phase. This way we divide the learning process into two components, leg swing manifold and support weight shift manifold - as the measure of optimality is essentially different for each phase.

We begin with the same neighbourhood graph computation procedure where we gradually increase the neighbourhood distance until the graph is not disconnected (Figure 18(b) and 18(a)). We set the dimensionality of the manifolds to be 4, with a simple cross validation step that penalizes model complexity while producing stable and reasonable results. In all learnt manifolds we used models with 20 RBFs, and 500 data points that belong to 25 random task space trajectories as described in the previous section.

3) Generation of novel walking solutions: The learnt manifolds are able to produce smooth walking trajectories that satisfy the optimization criteria used to produce the training data. Moreover, trajectories are produced approximately within *one to two seconds*, in contrast to the numerical optimization used to generate the data which required on average approximately *45 seconds* per trajectory (Figure 20), both with reasonable code and on commodity hardware. The computation time of the former increases with the dimensionality of the manifold.

The procedure is able to produce stable walking in the continuum of the reachable space of the robot as depicted in Figure 19(a) and 19(b) for right and left swings accordingly. One interesting observation is that the robot manufacturer in the accompanying software for walking, specifies that the stepping space of the feet cannot extend more than 9cm. With our manifold trajectory generation we are able to step further and reach stably up to 12cm, nonetheless most of our experimental sampling was constrained to be up to 10cm.

One point to note is that the shape of the generated trajectories in task space is qualitatively different from the training data. The training data is generated by point-by-point

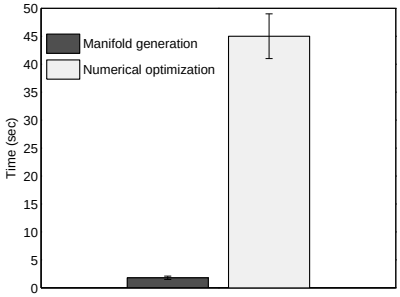


Fig. 20. Results averaged over 25 random start–to–end trajectories. The manifold representation can stepping trajectories that generalize from the numerical optimization examples in significantly less time. This way the manifold can been employed in an *online* scenario while producing trajectories that, with a numerical approach, would take close to a minute to compute.

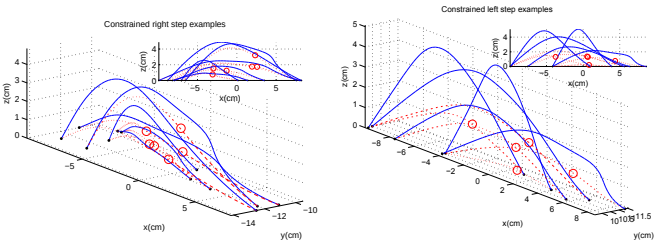


Fig. 22. Generated constrained task space trajectories from randomly sampled start and end points. The red trajectories correspond to the original unconstrained foot midpoint that collide with the obstacle (red circle). The resulting optimized constrained task space trajectories plotted in blue. Inset plots are side views of the identical trajectories. (Note that learning and generation of the geodesic trajectories takes place in the high dimensional joint space while the figures portray the outcome through the forward kinematics.)

kinematic optimization of an artificially imposed sinusoidal sequence of task space points. By fitting the tangent space of the manifold to the collection of all such data points, and making all local frames consistent, we extract a manifold that indeed traces the true underlying geometry that the optimization procedure sculpts in the robot joint space.

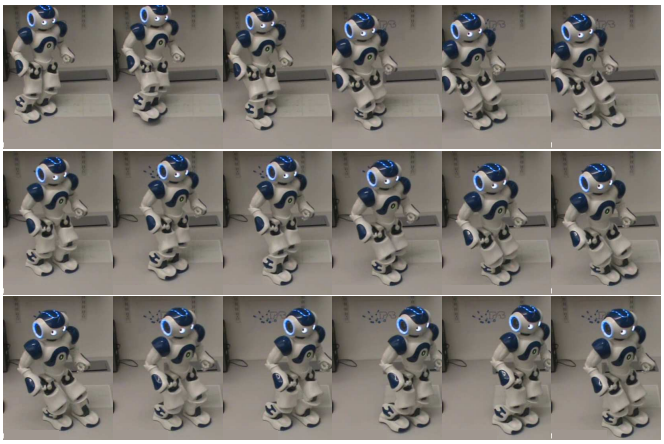


Fig. 21. An example of a generated walk from random feet start and end–point positions. The generated walk is stable and relatively fast while the possible reach of the steps is greater than the canned walking motions that comes preprogrammed with the robot. Additionally the motion generation is fast and can be employed in an online motion planner.

B. Constrained stepping with the Nao humanoid

Again, we focus on the task of walking, with the aim of generating a motion synthesis strategy that achieves full coverage of a reasonably large interval in step length, width and height. In effect, the optimality surface would be the set of all solutions to all possible task space queries, thus a tangent space point would have a local coordinate frame that guides the path in that particular neighborhood. We begin with a redundancy resolution strategy that would yield walking examples as training data for manifold learning.

1) Implementation: Identical to section VII-A2.

2) Generation of constrained walking motions: The learnt manifolds are able to produce smooth walking trajectories that satisfy the optimization criteria used to produce the training data. Moreover, trajectories are produced approximately within 1.5 *seconds*, in contrast to the numerical optimization used to generate the data which required on average approximately 45 *seconds* per trajectory, both with reasonable code and on commodity hardware. The computation time of the former increases with the dimensionality of the manifold.

We collect all the generated trajectories and compared with the ground truth data, i.e. the trajectories that the optimization procedure would have generated. We average over 50 trials and achieve an RMSE of 0.1 at a tiny fraction of the computational cost (2%). This way we can replace the computational expensive procedure with our manifold representation and be able to generate cheaply, equally accurate walking solutions.

As with the 3-link arm example we randomly pick a set of start and end points in task space, generate a trajectory as a geodesic path on the learnt skill manifold and insert an obstacle near the trajectory. Examples of this process are depicted on Figure 22 for right and left foot midpoint task space trajectories. Note that the dashed red lines correspond to the unconstrained predictions that collide with the perceived obstacles, that appear as red circles.

The efficacy of such an additional degree of control is obvious. To provide a concrete example we have used the constrained geodesic trajectory generation for random obstacle avoidance, staying away from regions in task space that might interfere with the swing trajectory. In the case of going up or down a step, it is often the case that the foot collides with the previous or next step's edge. When such a collision occurs, the robot loses its balance and falls down. Now, we can detect this collision and set this point to be a *"no-go"* point in a point set. In the same state the robot will then skillfully avoid the colliding pose and successfully negotiate the step. Snapshots of such behaviour are shown in Figure 23 and 24.

C. Standing on one leg with the Nao humanoid

Consider the task of standing on one leg while moving the other freely around. You will soon realize that balancing on one leg is not a trivial task while it will become clear that there is a certain area that your free leg can cover, after which stabilization efforts would be in vain. Capturing this region of stability of a humanoid system performing such a task is the focus of this section.

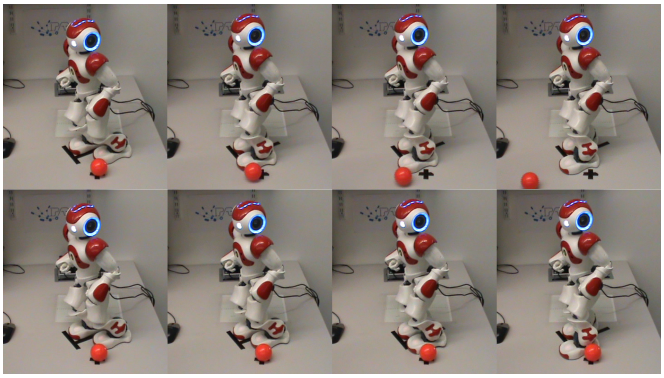


Fig. 23. Nao executing of a planned motion. *Top*; the unconstrained stepping trajectory that hits an obstacle (the ball). *Bottom*; the constrained optimized trajectory where the swing path avoids the obstacle, the ball.

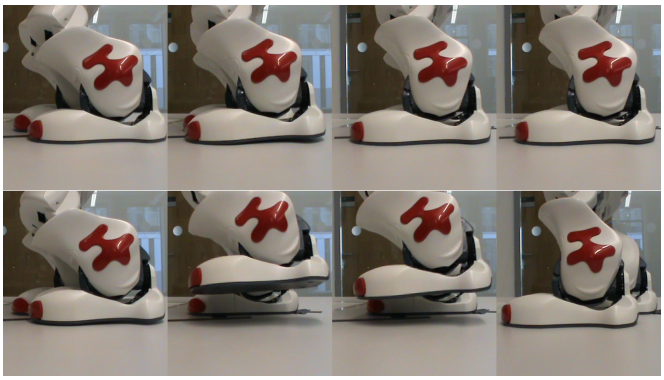


Fig. 24. Detail of a left foot swing. *Top*; the original trajectory that provides minimal foot clearance (approx. 2cm on apex). *Bottom*; adding a obstacle close to the original trajectory pushes the optimization to higher stepping trajectories (here approx. 5cm on apex).

We consider a more elaborate example with Nao humanoid robot. We show that by utilizing a learnt skill manifold we can capture the subspace of the configuration space of the system that consists of the set of poses that stably balance the humanoid on one leg. This skill manifold is learnt from a set of example data that we arrive at through numerical pose optimization of a variety of costs as explained later below. We show that we can successfully generate paths on such manifold as well as project random pose samples that are unstable to their closest stable pose, thus ensure the stability of the system.

1) Implementation: For this example we use the Nao humanoid robot and, as previously, we focus on the 12 DoFs of the lower body, hip and legs. These are the DoFs that are most relevant to the stability of the plant as the arms and the heads have little impact on the stability of the pose, even though their use can make a big difference in more dynamic situations.

We begin by randomly sampling points in the task space of the robot in the interval;

$$\mathbf{x} \quad = \quad \begin{cases} x \in [-20, 20] \\ y \in [-20, -5] \\ z \in [0, 15] \end{cases} \tag{24}$$

This covers a large volume of the reachable set of the free swing leg (Figure 25). Nonetheless this sampling produces points that are not reachable without taking an extra step

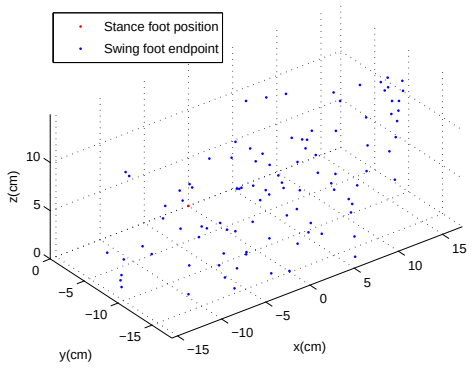


Fig. 25. Task space samples drawn from a standard uniform distribution on the interval in Eq. 24. Unreachable samples are discarded through the numerical optimization step. The red dot signifies the position of the stance foot while the blue dots represent the midpoint of the foot that moves freely.

TABLE II
EVALUATION TABLE. THE PROJECTED STATES RMSE IS EVALUATED AGAINST GROUND TRUTH DATA THAT IS GENERATED FROM THE NUMERICAL OPTIMIZATION PROCEDURE. RESULTS ARE AVERAGED OVER 50 TRIALS.

	Time	RMSE	*Unstable*	*Stable*
Numerical optimization	38.35*sec*	–	2	48†
Random state samples	–	–	50	0
Projected states	1.707*sec*	0.01	3	47‡
Geodesic trajectories	0.0254*sec*	–	8	42‡

† Trials for samples that did not converge where not included in the count.
† ‡ 12 of the 13 unstable poses pass model evaluation but fail on the physical plant due to self collision.

or might collide with the robot geometry. The former will be discarded after the optimization technique, explained in the next subsection. Weeding out the latter would require a collision detection step that is beyond the scope of the experiment. Many of such samples require the robot to reach close to the boundary of its stability and this is exactly what we are interested in capturing. Once a random task space sample is drawn we pass it to the numerical optimization method that would return an optimized pose. Each pose consists of the 12 DoFs for the legs that are optimized, plus the head and arm DoFs set to a standard constant position.

2) Numerical optimization: Mapping from a 3 dimensional task position to a 12 dimensional pose is a general redundancy resolution problem. Since the mapping is not unique we treat it as a constrained nonlinear optimization problem. We used the approach detailed in VII-A1, evaluating a number of cost factors J^n. The search space q is also subject to inequality constraints that keep all produced configurations within the joint limits of the humanoid.

The cost function is a mixture of task constraints and stability constraints. There are cases when the goal position is outside the reachable area of the humanoid or the optimization suffers from a bad local minima, leading to a solution pose far from the task specifications. In practice, bad samples are seldom encountered and are easily characterized by the final optimization cost. Optimization results that achieve a suboptimal final cost are discarded.

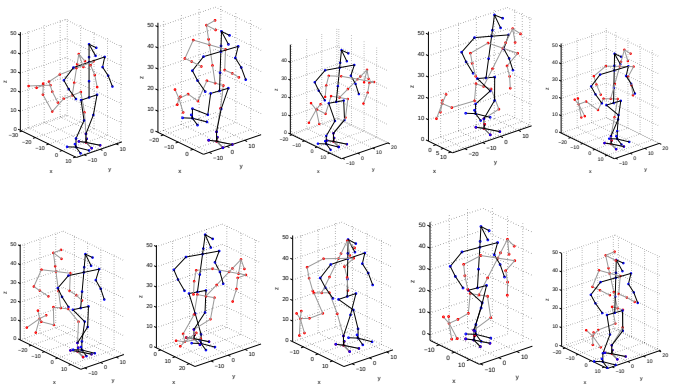


Fig. 26. The skeleton model of the Nao humanoid performing a number of poses that balance on one leg. The gray skeleton represents the random pose sample that has been sampled within the space of kinematic constraints of the system. The black skeleton is the projection of the previously randomly sampled pose on the stable configuration manifold. The snapshots on Figure 27 correspond to the same sequence of poses. Also available in the accompanying video.

3) Evaluation: The evaluation of the method is multi-fold. First, a good metric is to compare against the poses that the computational optimization method would produce. The error between such predictions and ground truth can give a reliable estimate of the accuracy of the approximation that the manifold method provides. This way we compute the RMSE of poses produced by the projection operation against the outcome of the numerical optimization given the same query. Averaged over 50 samples we achieve an RMSE of 0.01, meaning that our representation approximates the true underlying geometry closely.

Next, we compare the time that it would take to produce such predictions, through both the optimization process and the manifold projection, as well as how much time it would take to generate a full geodesic trajectory given the current and goal states. On average the numerical optimization procedure requires more than 35 seconds per point. Projecting a random point requires 1.7 seconds on average, depending on the initial estimate and step size. Producing a geodesic path given start and goal points requires just 0.02 seconds. All evaluations where carried out on commodity hardware with reasonable MATLAB code[5]. This points out the immediate benefit that one can enjoy from using such a skill manifold representation.

Finally, we evaluate the stability of the initial random poses, the subsequent projections of the random poses on the learnt manifold, and the geodesic paths that are produced to reach the sequence of poses. Almost all poses, except from the random state samples, are stable with a little variation between cases. In addition almost all unstable outcomes are due to self-collisions, something present in the training data, the removal of which is not the objective of our experiment. An overview of aforementioned evaluations is available in Table II.

Figure 26 and Figure 27 provide an example of state projection with the Nao humanoid. Random poses are generated in the system's state space. All these are unstable poses that

[5]Porting all code to *mex* files would in principle provide a great speed-up but is beyond the scope of this work.

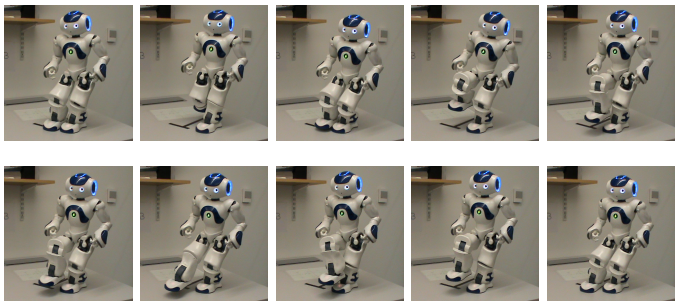


Fig. 27. Snapshots of the video that demonstrates how the Nao humanoid can move through the poses presented in Figure 26. The motions from one pose to the next is a geodesic path on the stable configuration manifold. Also available in the accompanying video.

are only restricted to be within the robots joint limits. These poses are then projected on the learnt manifold and a new pose q' is found. This pose is the closest state to the random state that belongs to the manifold hypersurface, i.e. q' is the projection of the random pose onto the manifold hypersurface. We then generate the geodesic trajectory that originates from the current pose of the robot to the projected state. A set of 10 such poses are demonstrated in Figure 26 while Figure 27 shows the poses on the physical Nao humanoid. A video of this sequence is also available as accompanying material.

Overall the evaluation shows that the manifold representation is a close approximation of the solution space that the demonstrated solution set derives from. The key advantage is that such an encoding has far superior computational efficiency while also allows the utilization all the tools that we have presented in previous sections. In essence it gives us the ability to arrive at novel solutions that are optimal with respect to the presented examples at computational cost that allows on-line deployment.

VIII. Comparison to Learning by Demonstration

In the preceding sections we proposed a framework for motion planning and control based on a manifold learning approach which encodes a solution set derived from example data in a model free manner. We have seen how such a representation can be used to, on the one hand, answer novel planning queries and, on the other hand, control the execution of planned trajectories on-line in a reactive fashion.

In this section we evaluate our framework against a state of the art imitation learning approach. We demonstrate that our approach provides superior generalization and stability characteristics, as well as a far greater ease of re-targeting, both with respect to initial and goal positions.

For each scenario we change the start position, or goal respectively, of the task by first a small and then a large offset. We evaluate both learning frameworks on these sets of novel planning queries. We provide evidence of consistently good behaviour of the trajectories generated by our manifold framework.

A. Learning by demonstration

Learning by Demonstration (LbD) is a state of the art *Imitation Learning* approach. LbD uses a statistical approach

to estimate the underlying dynamics of a, generally small (< 10), set of example trajectories. After the learning phase, the model is used to predict a velocity vector given the state of the system. This is in turn integrated to the system's state and the procedure is repeated until the state has converged to an attractor point of the approximated dynamics. The next subsection provides a more detailed overview of the LbD framework.

1) The LbD approach: LbD is a two phase framework, consisting of a leaning phase, carried out off-line, and a generative phase, that can be used on-line. The learning phase utilizes the Gaussian Mixture Model (GMM) approach to model the dynamics of the data generation process. Generation is done by Gaussian Mixture Regression (GMR), a procedure that generates samples from a learnt GMM. For a more detailed presentation of the GMR/GMM framework please refer to Gribovskaya et al. (2010).

The GMM/GMR framework allows for stability analysis and empirical determination of the region of stability of the learned dynamics. Disadvantages include the appearance of spurious attractors that can trap the evolution of the state of the system and the inversion of the covariance matrix that can suffer from singularities. Scaling up to systems with a high number of DoFs and complex dynamics is one of the central difficulties of the approach. For that most recent results consider task space encodings, limited to a 2 or 3 dimensional Cartesian space (Gribovskaya et al. (2010)), thus requiring an extra layer of inverse kinematics and dynamics that eventually computes the actual joint space trajectory that the system follows.

B. iCub data

The dataset used for the comparison of the two methods comes from the iCub humanoid robot[6]. It consists of 5 trajectories, discretized to 100 datapoints. The trajectories represent the end-effector position and orientation in Cartesian space. End-effector positions are $\mathbf{x} = [x, y, z]$, while the orientations are in quaternion notation, $\mathbf{o} = [o_1, o_2, o_3, o_4]$. In essence the dataset encodes the task dynamics and not the dynamics of the robot. The task in this case would be *pick-and-place* motions.

[6]The dataset and source code are available at http://lasa.epfl.ch/sourcecode/index.php, under *Learning Position and Orientation Control*.

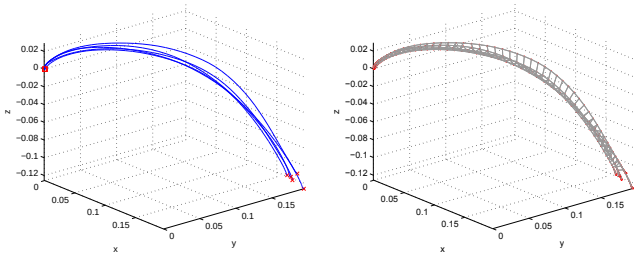


Fig. 28. (*Right*) iCub *pick-and-place* task. The data consists of 5 trajectories of 100 datapoints each. (*Left*) The neighbourhood graph that results from the iCub data.

TABLE III
EVALUATION OF BOTH METHODS WITH REGARDS TO TRAINING AND GENERATION TIMES. THE RESULTS ARE AVERAGED OVER MULTIPLE TRIALS (20 AND 27 RESPECTIVELY).

	Num. of kernels	*Training time*	*Generation time*
Manifold model	5	1.38*sec*	0.0817*sec*
GMM/GMR	5	2.35*sec*	0.0853*sec*

All five demonstration trajectories originate approximately around $\mathbf{x}_{start} = [1.8, 1.8, -1.1]$, while the goal position is the origin of the axis, $\mathbf{x}_{goal} = [0, 0, 0]$, as plotted on Figure 28.

C. Model comparison

For the comparison we have used the model provided along with the source code of the implementation. The GMM/GMR model divides the state of the system in two parts. This division is further requires two GMM/GMR models, one used for learning and predicting positions and another for learning and predicting orientations. The separation of position and orientation estimates is rather unnatural as, in tasks such as *pick-and-place*, these are strongly coupled. We focus our comparison to the positional part of the task as a poor prediction of position would automatically render any prediction in orientation futile. The position GMM consists of 5 Gaussian kernels that encode for a state model of the form:

$$\mathbf{x} = [x, y, z, \dot{x}, \dot{y}, \dot{z}]^T. \tag{25}$$

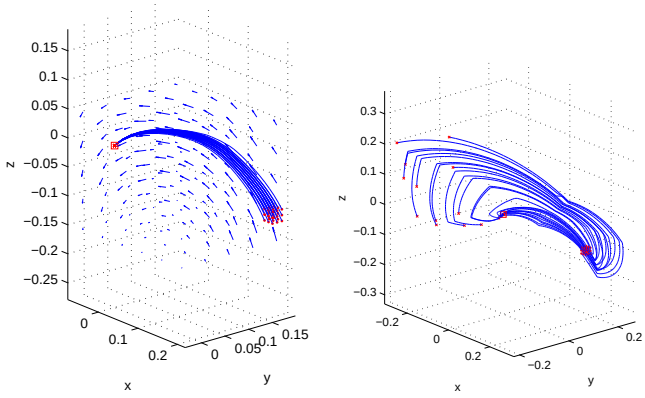
Each kernel, $G^{k=1,\dots,6}$, is thus parametrized by $\mu([6 \times 1])$ and $\sigma([6 \times 6])$, while the GMM model also contains a prior probabilities vector, π_k. The aforementioned parameters are set in the learning phase automatically. The GMM model takes approximately 2.35 seconds to train from the given data.

For the manifold model we use the position data as described earlier. In fairness for the comparison we set the number of RBFs, k, to 5, but in practice we achieve an acceptably low model error with 4 kernels. The dimensionality of the tangent basis, d, is set to 1 in a *cross-validation* manner, as this is sufficient to explain the dataset at hand.

We begin with the calculation of the nearest neighborhood relationships between the datapoints. The constraint that we impose is that all datapoints should belong to a single connected component and that all consequent datapoints for each trajectory are connected (temporal relation). The neighborhood graph that results from this operation is available on Figure 28. Next we learn a manifold model, $\mathcal{M}$, with the procedure described in section III-B. The computation of the neighborhood graph as well as the manifold learning, requires approximately 1.38 seconds to complete.

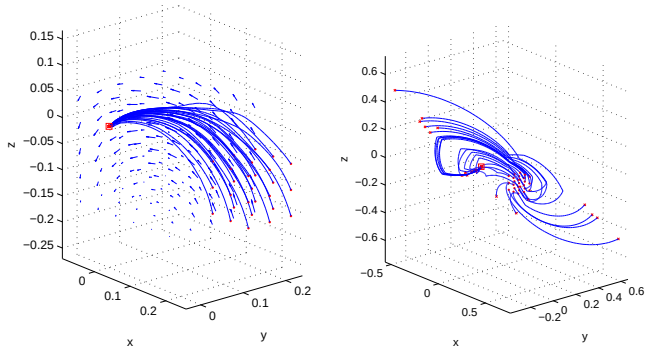
D. Results

Having compared the two models on training and generation times, we now compare the actual trajectories that the two methods generate. We begin by comparing the predicted trajectories under a change of the starting position. To do so we set up a a cube centered at the point chosen in the GMM/GMR



(a) Manifold. (b) GMM/GMR.

Fig. 29. Comparison of the two methods by altering the starting positions of the trajectories. The novel starting positions are chosen on a small ($\ell = 0.02$) cube around a start point of one demonstrated trajectories. (*a*) Generated trajectories from the manifold model. All (27) trajectories reach the target in a manner qualitatively similar to the demonstration data. The blue arrows are evaluations of the tangent space in the ambient space of the manifold. (*b*) Trajectories generated from the GMM/GMR model. Only 12 trajectories reach the target position, while the rest get trapped to flows of spurious attractors and fail to converge.



(a) Manifold. (b) GMM/GMR.

Fig. 30. Comparison of the two methods by altering the starting positions of the trajectories. The novel starting positions are chosen on a large ($\ell = 0.1$) cube around a start point of one demonstrated trajectories. (*a*) Generated trajectories from the manifold model. All (27) trajectories reach the target in a manner qualitatively similar to the demonstration data. The blue arrows are evaluations of the tangent space in the ambient space of the manifold. (*b*) Trajectories generated from the GMM/GMR model. Only 8 trajectories reach the target position, while the rest get trapped to flows of spurious attractors and fail to converge.

code as the starting position. This point is the initial position of one of the demonstrated trajectories.

We run two tests on starting positions; one utilizing a cube of $0.02m$ edges (*small cube*) and one where the edges of the cube measure $0.1m$ (*large cube*). We subsequently set as starting positions the midpoints of the edges of the cube and the points that are on the corners of each edge. This results to a set of 27 points that are equally spaced with regards to the initial point as,

$$\mathbf{x}_{cube} = \begin{cases} x_{init} \pm 0.01 \\ y_{init} \pm 0.01 \\ z_{init} \pm 0.01 \end{cases} (small\ cube), \tag{26}$$

$$\mathbf{x}_{cube} = \begin{cases} x_{init} \pm 0.05 \\ y_{init} \pm 0.05 \\ z_{init} \pm 0.05 \end{cases} (large\ cube). \tag{27}$$

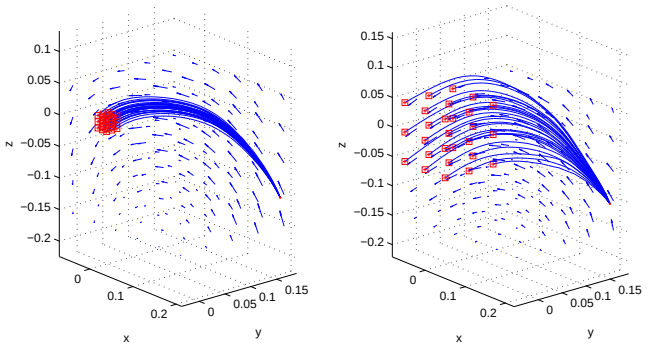
For both cases the target of the trajectories is set to $x_{goal} = [0, 0, 0]$. We run the set of 27 start/end tuples through the GMM/GMR and manifold trajectory generation procedures, both for the small and large cube starting positions. The results of the small cube staring position changes are show in Figure 29, while the large cube results are plotted in Figure 30. The varying initial positions are marked with red points, the points that each trajectory reaches are red $\times$s, and the goal position is marked with a red square.

The outcomes of this process are plotted in Figure 29 and Figure 30. We see that in both scenarios the manifold model can successfully generate trajectories that reach the goal position. What is more, the generated trajectories are *qualitatively* very similar to the demonstrated trajectories. The trajectories that the GMM/GMR framework generates are very sensitive to the change of the starting position. This way, in both scenarios, most of the trajectories quickly fall under the influence of spurious attractors and are subsequently attracted away from a desired evolution. This effect becomes stronger when we alter the initial position according to the large cube points (Figure 30(b)). We attribute this effect to the tight fit of the GMM around the demonstrated data, resulting to this poor generalization performance. In contrast the manifold trajectory generation yields very good results, even when we initialize from the points on the large cube (Figure 30(a)). We see that the generated trajectories reach the goal position successfully while also maintain a spatial profile very similar to the demonstrated trajectories.

Next we present results from the manifold method with respect to changes of the goal position. We follow the same approach of creating a small and a large cube around the goal position and picking points from these geometries. Changing the goal is a straightforward procedure within our manifold framework. In contrast, changing the goal of a GMM/GMR model is not intuitive. The reason is that the GMM model is grounded on the state space that it represents in an absolute manner. Thus, a change of goal would require a translation of the full model in state space, something that would result in the former initial positions being outside the volume that the model covers. Intuitively, one would need to scale and translate the GMM model with the new goals in mind. This becomes increasingly difficult as the state space contains position and velocity variables, the coupling of which is sensitive to scaling. Resolving the model scaling problem is beyond the scope of this paper, thus we present examples of goal changes only for the manifold model.

The results of this experiment are available on Figure 31, both for small and large changes of the goal position. These show that first; it is easy to change the goal of a planning query and second; the manifold encoding produces trajectories that exhibit spatial profiles that are qualitatively very similar to the original set of examples. A summary of all the trials is available on Table IV.

1) Manifold metric: It is also interesting to investigate what the manifold metric, that we introduced in section IV, would



(a) Small change. (b) Large change.

Fig. 31. Change of the goal positions for the manifold generation procedure. (*a*) Generated trajectories that reach points on a cube of edge ($\ell = 0.02$) around the former goal position. (*b*) Generated trajectories that reach points on a cube of edge ($\ell = 0.1$) around the former goal position. The generated trajectories demonstrate a spatial profile that is very similar to the demonstrated data.

TABLE IV

A SUMMARY OF THE COMPARISON BETWEEN THE MANIFOLD AND THE GMM/GMR FRAMEWORKS. WITH THE MANIFOLD REPRESENTATION ALL THE TRAJECTORIES SUCCEEDED IN REACHING THE GOAL POSITION. IN THE GMM/GMR CASE TRAJECTORIES PROVED TO BE VERY SENSITIVE TO CHANGES OF THE INITIAL POSITION, LEADING TO POOR PERFORMANCE. WHAT IS NOT CAPTURED BY THE TABLE IS THE QUALITATIVE BEHAVIOUR OF THE GENERATED TRAJECTORIES, THAT IN THE CASE OF THE MANIFOLD ENCODING, IS VERY SIMILAR TO THE DEMONSTRATED EXAMPLES.

		Manifold		GMM/GMR	
		Success	*Failure*	*Success*	*Failure*
Change of *start*	*Small*	27	0	12	15
	Large	27	0	8	19
Change of *goal*	*Small*	27	0	–	–
	Large	27	0	–	–

result in with respect to the demonstrated iCub dataset. In such a setting this metric would provide us with the means necessary to perform online feedback control when executing a planned pick and place trajectory. In essence the metric would give us a quantitative estimate of the desirability of the states that populate the off-manifold state space.

When executing a pick-and-place trajectory that is generated by the manifold representation, as a geodesic path, whenever an unforeseen perturbation occurs we can quickly compute the closest-on manifold state and reactively generate a trajectory to it. This additional benefit of the manifold representation can be interpreted as a state value, or cost, that forms a "desirable" tunnel that surrounds the demonstrated trajectories and can be directly used for reactive feedback control.

Figure 32 presents a volumetric plot of the manifold metric. The blue lines are the set of demonstrated trajectories, originating from the red $\times$s and reaching the goal state marked by a red square. The distance to the manifold is color-coded, red being the most distant states and the distance decreasing towards the blue volume. States that are closer to the manifold are transparent for clarity. The three Figures are rotated views of the same plot. Figure 32 demonstrates how the manifold metric creates a tunnel of desirable states that surround the demonstrated solution set.

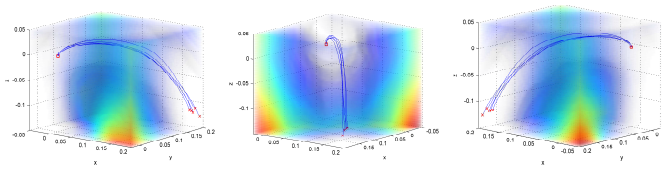


Fig. 32. Volumetric plot of the manifold metric that allows for reactive feedback control. The color-coding signifies distance to the manifold model, ranging from red (furthest) to blue. Small distance volumes are transparent for clarity. All three figures are rotated views of the same plot and demonstrate how the manifold metric creates a tunnel of desirable states that surround the demonstrated solution set.

IX. CONCLUSION

We address the problem of encoding skills, including constraints and variability implicit in the set of feasible solutions to an underlying problem (which is typically akin to an optimal control problem), and using this encoding for the purposes of trajectory generation in a constrained setting, such as when novel obstacles must be accounted for. A key feature of our approach is that we naturally cope with incomplete initial specifications in that we do not assume an analytical task specification at the outset. Instead, we learn from data (which may come from traces of a human expert of a collection of numerical optimization solutions to instances of the problem class) to induce a manifold encoding the skill and we devise algorithms for motion planning on this manifold and reactive control when the system is perturbed, either along the manifold or, more importantly, away from the manifold. We discuss many different experiments to elucidate the concepts, including comparative experiments against a state of the art imitation learning method - to identify weaknesses of existing approaches that we are able to circumvent.

We build on many different lines of work. Control theorists, especially in the area of geometric control theory, have long investigated ways to exploit inherent symmetries in dynamical systems to define control strategies. However, often, these results have been restricted to a few specific types of systems and have not yet addressed the rich variety of robotic motions we wish to implement with modern humanoid systems. In machine learning, manifold learning is becoming an established method, although mostly used to define reduced dimensional coordinates within which visualization or clustering procedures are devised. As we have noted above, this is quite different from the objective of planning and control where we need to be able to enforce corrective vector fields in the ambient space while simultaneously utilizing a low-dimensional task encoding. This has been the focus of our work and this viewpoint and use of the underlying tool in a novel way is an aspect of our novel contributions.

There are many ways in which one could build on the ideas described above. We have demonstrated our approach on a number of different robot platforms, to argue for the broader applicability of the methods. Our approach applies to skills of varying dimensionality though its computational efficiency is inversely proportional to the inherent manifold dimensionality. The class of problems that our approach can best address

is that of robotic skills that produce complex but smoothly varying solution sets, i.e. do not exhibit discontinuous changes of behaviour. However, we have restricted attention to motion planning in configuration spaces, involving manifolds in a state space without higher order terms such as velocities and accelerations. Incorporating these would be of great interest as we target more dynamic movements. Also, we have presented one way to define a corrective vector field in the ambient space, to compensate for large perturbations. Our construction is based on the fact that we do not have an explicit task specification apart from the demonstration traces, so that one natural solution to the problem of perturbations is to quickly revert to the feasible subspace. There are clearly going to be cases where this is suboptimal and a better corrective strategy is possible. Devising such strategies despite incomplete task specifications should be of great practical interest within robotics. More generally speaking, the solution presented in this paper pertains to one skill, such as stably standing on one foot. A competent autonomous robot must marshal many such skills in order to function effectively in the human-competitive manner we roboticists often envision. This requires methods to compose skill manifolds, to learn suitable compositions and to learn to perform motion synthesis in this extended setting. We anticipate, and hope, that even in this extended setting, encodings such as ours could provide both conceptual and computational benefits.

Acknowledgment

This work has taken place in the Robust Autonomy and Decisions group within the School of Informatics. Research of the RAD Group is supported by the UK Engineering and Physical Sciences Research Council (grant number EP/H012338/1) and the European Commission (TOMSY Grant Agreement 270436, under FP7-ICT-2009.2.1 Call 6).

References

Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, New York, NY, USA, 2004. ACM.

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.

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.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1st ed. 2006. corr. 2nd printing edition, October 2007. ISBN 0387310738.

Sebastian Bitzer, Ioannis Havoutis, and Sethu Vijayakumar. Synthesising novel movements through latent space modulation of scalable control policies. In *Lecture Notes in Computer Science*, pages 199–209. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-69133-4. doi: 10.1007/978-3-540-69134-1_20.

V. Boor, M.H. Overmars, and A.F. van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. *IEEE International Conference onRobotics and Automation*, 2, 1999.

Matthew Brand. Charting a manifold. In *Advances in Neural Information Processing Systems 15*, pages 961–968. MIT Press, 2003.

Tim Bretl, Sanjay Lall, Jean-Claude Latombe, and Stephen Rock. Multi-step motion planning for free-climbing robots. In *in Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 1–16, 2004.

R. R. Burridge, A. A. Rizzi, and D. E. Koditschek. Sequential Composition of Dynamically Dexterous Robot Behaviors. *International Journal of Robotics Research*, 18(6):534–555, 1999.

S. Calinon and A. Billard. A probabilistic programming by demonstration framework handling skill constraints in joint space and task space. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '08)*, pages 367–372, September 2008.

S. Calinon and A. Billard. Statistical learning by imitation of competing constraints in joint space and task space. *Advanced Robotics*, 23:2059–2076, 2009.

R. Chalodhorn, D.B. Grimes, G.Y. Maganis, R.P.N. Rao, and M. Asada. Learning humanoid motion dynamics through sensory-motor mapping in reduced dimensional spaces. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '06)*, May 2006.

David C. Conner. *Integrating Planning and Control for Constrained Dynamical Systems*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January 2008.

David C. Conner, Alfred Rizzi, and Howie Choset. Composition of local potential functions for global robot control and navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, volume 4. IEEE, October 2003.

David C. Conner, Howie Choset, and Alfred Rizzi. Integrated planning and control for convex-bodied nonholonomic systems using local feedback. In *Proceedings of Robotics: Science and Systems II*, pages 57–64, Philadelphia, PA, August 2006. MIT Press.

D.C. Conner, H. Choset, and A.A. Rizzi. Flow-through policies for hybrid controller synthesis applied to fully actuated systems. *IEEE Transactions on Robotics*, 25, 2009.

Jose Costa and Alfred O. Hero. Manifold learning with geodesic minimal spanning trees. *Computing Research Repository (CoRR)*, 2003.

J. J. Craig. *Introduction to Robotics.* Addison Wesley, 2nd edition, 1989.

Chris Dever, Bernard Mettler, Eric Feron, Jovan Popovic', and Marc Mcconley. Trajectory interpolation for parametrized maneuvering and flexible motion planning of autonomous vehicles. *AIAA Guidance, Navigation, and Control Conference*, 2004.

Chris Dever, Bernard Mettler, Eric Feron, Jovan Popovic', and Marc Mcconley. Nonlinear trajectory generation for

autonomous vehicles via parameterized maneuver classes. *Journal of Guidance, Control and Dynamics*, 29:289–302, 2006.

Rosen Diankov, Nathan Ratliff, David Ferguson, Siddhartha Srinivasa, and James Kuffner. Bispace planning: Concurrent multi-space exploration. In *Robotics: Science and Systems*, June 2008.

P. Dollár, V. Rabaud, and S. Belongie. Learning to traverse image manifolds. In *Neural Information Processing Systems (NIPS)*, Dec. 2006.

P. Dollár, V. Rabaud, and S. Belongie. Non-isometric manifold learning: Analysis and an algorithm. In *International Conference on Machine Learning (ICML)*, June 2007.

Brian Eriksson and Mark Crovella. Estimating intrinsic dimension via clustering. In *IEEE Statistical Signal Processing Workshop (SSP)*, Ann Arbor, MI, August 2012. URL http://www.cs.bu.edu/faculty/crovella/paper-archive/ssp12-cluster-dimension.pdf.

E. Frazzoli, M. A. Dahleh, and E. Feron. A maneuver-based hybrid control architecture for autonomous vehicle motion planning,. In T. Samad and G. Balas, editors, *Software Enabled Control: Information Technology for Dynamical Systems*. Wiley-IEEE Press, 2003.

E. Frazzoli, M. A. Dahleh, and E. Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Trans. on Robotics*, 21(6):1077–1091, December 2005.

RJ Full and DE Koditschek. Templates and anchors: neuromechanical hypotheses of legged locomotion on land. *Journal of Experimental Biology*, 202(23):3325–3332, 1999. URL http://jeb.biologists.org/cgi/content/abstract/202/23/3325.

Elena Gribovskaya, Khansari Zadeh, Seyed Mohammad, and Aude Billard. Learning Nonlinear Multivariate Dynamics of Motion in Robotic Manipulators. *International Journal of Robotics Research*, 2010. ISSN 0278-3649. doi: NA.

T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, August 2001. ISBN 0387952845.

Ioannis Havoutis and Subramanian Ramamoorthy. Geodesic trajectory generation on learnt skill manifolds. *International Conference on Robotics and Automation (ICRA), 2010. Proceedings 2010 IEEE*, 15-19, 2010a.

Ioannis Havoutis and Subramanian Ramamoorthy. Constrained geodesic trajectory generation on learnt skill manifolds. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '10).*, October 2010b.

Matthias Hein and Jean-Yves Audibert. Intrinsic dimensionality estimation of submanifolds in rd. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 289–296, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5. doi: 10.1145/1102351.1102388. URL http://doi.acm.org/10.1145/1102351.1102388.

M. Hersch, F. Guenter, S. Calinon, and A. Billard. Dynamical system modulation for robot learning via kinesthetic demonstrations. *IEEE Transactions on Robotics*, 24(6):1463 –1467, dec. 2008. ISSN 1552-3098. doi: 10.1109/TRO.2008.2006703.

A.J. Ijspeert, J. Nakanishi, and S. Schaal. Trajectory formation for imitation with nonlinear dynamical systems. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 2, pages 752 –757 vol.2, 2001. doi: 10.1109/IROS.2001.976259.

Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *In IEEE International Conference on Robotics and Automation (ICRA '02)*, pages 1398–1403, 2002.

P. Isto and M. Saha. A slicing connection strategy for constructing prms in high-dimensional cspaces. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '06).*, pages 1249–1254, May 2006. ISSN 1050-4729. doi: 10.1109/ROBOT.2006.1641880.

Odest Chadwicke Jenkins and Maja J Mataric. A spatio-temporal extension to isomap nonlinear dimension reduction. In *International Conference on Machine Learning (ICML '04)*, pages 441–448, 2004.

Balázs Kégl. Intrinsic Dimension Estimation Using Packing Numbers. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.3428.

M. B. Kobilarov and J. E. Marsden. Discrete geometric optimal control on lie groups. *IEEE Transactions on Robotics*, 27(4):641 –655, Aug. 2011. ISSN 1552-3098. doi: 10.1109/TRO.2011.2139130.

James J. Kuffner, Satoshi Kagami, Koichi Nishiwaki, Masayuki Inaba, and Hirochika Inoue. Dynamically-stable motion planning for humanoid robots. *Auton. Robots*, 12 (1):105–118, 2002. ISSN 0929-5593. doi: http://dx.doi.org/10.1023/A:1013219111657.

J.J. Kuffner and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '00)*, 2:995–1001 vol.2, 2000. doi: 10.1109/ROBOT.2000.844730.

S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

Steven M. LaValle and James J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001. doi: 10.1177/02783640122067453. URL http://ijr.sagepub.com/cgi/content/abstract/20/5/378.

Elizaveta Levina and Peter J. Bickel. Maximum Likelihood Estimation of Intrinsic Dimension. In *NIPS*, 2004.

William S. Levine. *The Control Handbook*. IEEE PRESS, 1996.

Andrew D. Lewis. Is it worth learning differential geometric methods for modeling and control of mechanical systems? *Robotica*, 25:765–777, November 2007. ISSN 0263-5747. doi: 10.1017/S0263574707003815. URL http://portal.acm.org/citation.cfm?id=1317140.1317151.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal. Learning and generalization of motor skills by learning from demonstration. In *International Conference on Robotics and Automation (ICRA '09)*, 2009. URL http://www-clmc.usc.edu/publications/P/pastor-ICRA2009.pdf.

S. Ramamoorthy and Benjamin J. Kuipers. Trajectory gen-

eration for dynamic bipedal walking through qualitative model based manifold learning. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 359–366, May 2008. ISSN 1050-4729. doi: 10.1109/ROBOT.2008.4543234.

Subramanian Ramamoorthy and Benjamin J. Kuipers. Qualitative hybrid control of dynamic bipedal walking. In *Proceedings of Robotics: Science and Systems*, Philadelphia, USA, August 2006.

Carl E. Rasmussen and Christopher Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. URL http://www.gaussianprocess.org/gpml/.

S. Rodriguez, Xinyu Tang, Jyh-Ming Lien, and N.M. Amato. An obstacle-based rapidly-exploring random tree. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '06)*, pages 895–900, 15-19 2006. ISSN 1050-4729.

S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500): 2323–2326, Dec 2000. doi: 10.1126/science.290.5500.2323.

Alla Safonova, Jessica K. Hodgins, and Nancy S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions in Graphics*, 23(3):514–521, 2004. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/1015706.1015754.

Stefan Schaal, Auke Ijspeert, and Aude Billard. Computational approaches to motor learning by imitation. *Philosophical Transactions: Biological Sciences*, 358(1431):537–547, 2003. ISSN 09628436.

Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer, 2008. ISBN 978-3-540-23957-4.

Robert F. Stengel. *Optimal control and estimation*. John Wiley & Sons, 2nd edition, 1995.

M. Stilman. Task constrained motion planning in robot joint space. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '07)*, pages 3074–3081, 29 2007-Nov. 2 2007. doi: 10.1109/IROS.2007.4399305.

Russ Tedrake, Ian R. Manchester, Mark Tobenkin, and John W. Roberts. Lqr-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29(8):1038–1052, 2010. doi: 10.1177/0278364910369189. URL http://ijr.sagepub.com/content/29/8/1038.abstract.

Yee Whye Teh and Sam Roweis. Automatic alignment of local representations. In *Advances in Neural Information Processing Systems 15*, pages 841–848. MIT Press, 2003.

J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, Dec 2000. doi: 10.1126/science.290.5500.2319.

S. Thomas, M. Morales, Xinyu Tang, and N.M. Amato. Biasing samplers to improve motion planning performance. *IEEE International Conference on Robotics and Automation (ICRA '07)*, pages 1625–1630, April 2007. ISSN 1050-4729. doi: 10.1109/ROBOT.2007.363556.

Emanuel Todorov and Weiwei Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *In proceedings of the American Control Conference, pp 300-306*, 2005.

Jakob Verbeek. Learning nonlinear image manifolds by global alignment of local linear models. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 28(8):1236–1250, aug 2006. URL http://lear.inrialpes.fr/pubs/2006/Ver06".

Sethu Vijayakumar, Aaron D'Souza, and Stefan Schaal. Incremental online learning in high dimensions. *Neural Compututation*, 17(12):2602–2634, 2005. ISSN 0899-7667. doi: http://dx.doi.org/10.1162/089976605774320557.

Jack M. Wang, David J. Fleet, and Aaron Hertzmann. Gaussian process dynamical models for human motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):283–298, 2008. ISSN 0162-8828. doi: http://dx.doi.org/10.1109/TPAMI.2007.1167.

Gu Ye and Ron Alterovitz. Demonstration-guided motion planning. *International Symposium on Robotics Research (ISRR)*, August 2011.

L. Zhang, S. LaValle, and D. Manocha. Global vector field computation for feedback motion planning. In *IEEE International Conference on Robotics and Automation (ICRA '09)*, pages 477–482, 2009.

APPENDIX A
INDEX TO MULTIMEDIA EXTENSIONS

Extension	Type	Description
1	Video	Examples of reactive control with learnt skill manifolds. Examples of manifold projection and path planning.