Motion Planning and Reactive Control on Learnt Skill Manifolds

Ioannis Havoutis

Doctor of Philosophy
Institute of Perception, Action and Behaviour
School of Informatics
University of Edinburgh
2011

Abstract

We propose a novel framework for motion planning and control that is based on a manifold encoding of the desired solution set. We present an alternate, model-free, approach to path planning, replanning and control. Our approach is founded on the idea of encoding the set of possible trajectories as a skill manifold, which can be learnt from data such as from demonstration.

We describe the manifold representation of skills, a technique for learning from data and a method for generating trajectories as geodesics on such manifolds. We extend the trajectory generation method to handle dynamic obstacles and constraints. We show how a state metric naturally arises from the manifold encoding and how this can be used for reactive control in an on-line manner.

Our framework tightly integrates learning, planning and control in a computationally efficient representation, suitable for realistic humanoid robotic tasks that are defined by skill specifications involving high-dimensional nonlinear dynamics, kinodynamic constraints and non-trivial cost functions, in an optimal control setting. Although, in principle, such problems can be handled by well understood analytical methods, it is often difficult and expensive to formulate models that enable the analytical approach.

We test our framework with various types of robotic systems – ranging from a 3-link arm to a small humanoid robot – and show that the manifold encoding gives significant improvements in performance without loss of accuracy. Furthermore, we evaluate the framework against a state-of-the-art imitation learning method. We show that our approach, by learning manifolds of robotic skills, allows for efficient planning and replanning in changing environments, and for robust and online reactive control.

Acknowledgements

I am grateful to my supervisor, Subramanian Ramamoorthy, for his invaluable guidance, advice and support throughout the course of my PhD. He has inspired me to work on manifold concepts and geometric approaches to robotics problems, while keeping me focused and motivated.

I wish to thank my second supervisor, Sethu Vijayakumar, for his valuable advice and for offering me the opportunity to undertake this PhD. He along with the other members of the SLMC group have provided me with a very stimulating research environment.

I would also like to thank my lab mates and the rest of the RAD group members. Last, a big thanks to my family and all friends that helped in making these years a delightful experience.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Ioannis Havoutis*)

To my family.

Table of Contents

List of Notation

Below is a list of symbols and abbreviations used throughout this thesis (unless an exception is noted in the text) . Entries of the form $a(\cdot)$ denote an argument should be supplied to the function a, for example where there is a direct dependency on some quantity. In addition to the terms defined here, note that we use the convention of bold upper-case letters, $\mathbf{A}$, to denote matrices, bold lower-case letters, $\mathbf{a}$, to denote vectors and normal weighted font, a, to denote scalar terms.

Symbols

$\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$	Position, velocity and acceleration in joint space.
$\mathbf{Q}(\cdot)$	Trajectory in joint space, either time indexed or discretised.
$\mathbf{x}, \dot{\mathbf{x}}, \ddot{\mathbf{x}}$	Position, velocity and acceleration in task space.
$\mathbf{X}(\cdot)$	Trajectory in task space, either time indexed or discretised.
t	Time.
T	Duration in time (e.g., duration of a trajectory).
$u(\cdot)$	Control action.
K_*	Gain (e.g., K_p, proportional gain).
D	Dimensionality of a high-dimensional space.
d	Dimensionality of a low-dimensional space.
$\mathcal{M}$	Manifold.
$\mathcal{H}(\cdot)$	Mapping from a point to its the tangent basis.

$\mathcal{H}(\cdot)$	Mapping from a point to its the tangent basis.
$\Delta^{i}_{\cdot j}$	Centred estimate of the directional derivative of i w.r.t. j.
ε^{ij}	Alignment factor.
θ	Vector of model parameters.
λ	Regularisation term.
$min(\cdot)$	Minimization of the argument construct.
μ	Mean, expected or arithmetic.
σ	Variance.
$\mathbf{A} \otimes \mathbf{B}$	The Kronecker product of the matrices $\mathbf{A}$ and $\mathbf{B}$.
$vec(\cdot)$	The vector operation that vectorizes a matrix $\mathbf{A}$.
O	Set of obstacle points.
f^{q}_{ij}	Force between consecutive path points q_i and q_j.
f^{O}_{ik}	Force between path point q_i and obstacle point O_k.
$\{\oslash\}$	Empty set.
C	Curvature of a set of (consecutive) path points.
$H'(\cdot)$	Manifold projection matrix.
$P(\cdot,\cdot)$	Joint probability density function.
G	Gaussian kernel.
Σ	Covariance of Gaussian kernel.

Abbreviations

SPL	Standard Platform League.
RRT	Rapidly-exploring Random Tree.
LbD	Learning by Demonstration.
PbD	Programming by Demonstration.
DoF	Degree(s) of Freedom.
LQR	Linear Quadratic Regulator.
LQG	Linear Quadratic Gaussian.
RBF	Radial Basis Function.
GMM	Gaussian Mixture Model.
GMR	Gaussian Mixture Regression.
PDF	Probability Density Function.
EM	Expectation Maximization.
DMP	Dynamic Motion Primitive.
RM	Road Map.
PRM	Probabilistic Road Map.
GPDM	Gaussian Process Dynamic Model.
GPLVM	Gaussian Process Latent Variable Model.
MDS	Multi-Dimensional Scaling.
LLE	Locally Linear Embedding.
SOM	Self-Organizing Map.
CFA	Coordinated Factor Analysis.
LLC	Locally Linear Coordination.

LSML	Locally Smooth Manifold Learning.
RMSE	Root Mean Squared Error.
NN	Nearest Neighbour.
ARA*	Any-time Repairing A*.
GP	Gaussian Process.

Chapter 1

Introduction

In recent years the *RoboCup* Robotic Soccer competition has become increasingly popular as an international event that brings together hardware and software engineering research in robotic soccer. One of the most successful divisions in the competition is the *Standard Platform League* (SPL). Each SPL team consists of a group of four *Nao* humanoid robots (Figure 1.1) that *autonomously* compete in matches of robotic soccer.

According to the official rule set, all SPL teams must have a designated Nao robot as a goalkeeper. As in human football, the Nao goalkeeper is charged with directly preventing the opposing team from scoring. Setting aside the vision requirements, successful goalkeeping requires intercepting shots at goal, basic ball handling and passing to a team mate further up the field. Such a small set of high level skill specification can be further broken down to a lower level skill set. Examples of such low level skills are walking and running to the ball, kicking to targets, diving for saves, etc.

The most common approach amongst current SPL teams is to have a small number of *hand-crafted* variations of each skill, that the high-level behaviour program would chose to replay at will. Such motion alphabets often lack feedback closure, thus are not robust to changes in the environment, while designing appropriate motions is time-consuming and subjective. This approach leads to discontinuous sets of action possibilities, largely impacting the efficiency of a football playing robot. For example, passing or kicking is often discretised to straight, left and right shooting motions, providing a very coarse control over the actual outcome of the action.

The Nao goalie example highlights the increasing need of modern, every-day, robotic platforms' requirement for real-time motion planning and control. Many realistic humanoid robotic tasks are defined by skill specifications involving high-dimensional nonlinear dynamics, kinodynamic constraints and non-trivial cost functions, in an op-

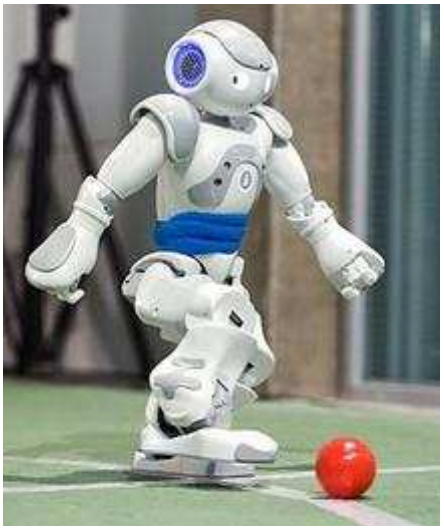


Figure 1.1: The *RoboCup Standard Platform League* Nao humanoid robot. {*Image courtesy of Team Edinferno*}.

timal control setting. Although, in principle, such problems can be handled by well understood analytical methods, it is often difficult and expensive to formulate models that enable the analytical approach.

Coming back to the SPL example, what we need is a flexible motion planning and control framework that can generate motions derived from a set of skills, necessary for successful goalkeeping. Such a motion generator should be able to achieve a large family of goals, for example it should be able to produce kicking motions that cover the entire space reachable by the robot's leg. In addition, it should take into account constraints such as joint limits, that arise from the physical grounding of the system, and task limits, that arise from the interaction with a dynamically changing environment, e.g., stability, obstacles, other opponents.

In a classical *optimal control* setting, e.g. Todorov and Li (2005), one would start with the dynamics modelling of the problem. This requires accurate analytical models of both the robot's kinematics and dynamics, as well as an accurate model of the environment and the interaction between the aforementioned parts, e.g. impact models, friction models. Such models can be difficult to acquire and can exhibit great variability, for instance different stepping surfaces, different motor behaviours, etc. By having a well defined goal and the analytical model, through an optimal control method, one can get to the optimal trajectory for achieving the goal at hand, alongside with local feedback stabilizers to overcome disturbances and perturbations. In practice such methods are very hard to apply to real world problems.

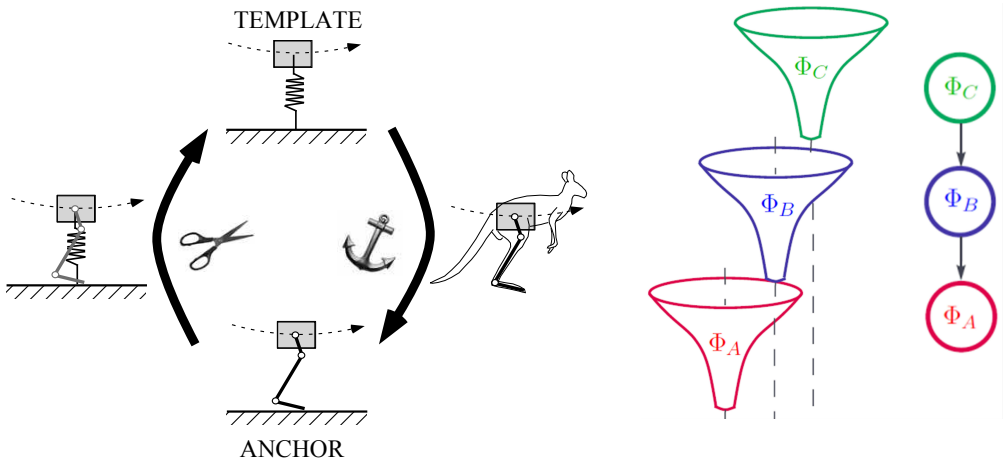
Although optimal control theory exits for a long time (Pontryagin, 1962; Athans and Falb, 1966), apart from a number of highly specialised applications, we are unable to use such methods in a day-by-day basis on systems of interest (e.g., multi-DoF, dynamic, humanoids), as the level of computational complexity quickly becomes prohibitive. The core of the problem is that when searching through all the possible trajectories that a system can generate, then this search space quickly grows too large. For example consider passing the ball to a team member with a directed kick. The motion generator need not search though any of the possible trajectories that, for example, do not move the kicking leg as such trajectory candidates would fail the task by default. In practice there is no need to generate every possible trajectory, we need to search through only those trajectories that capture some *structure* underlying a *specific* task.

One good approach, that can bring down the computational complexity, is to exploit qualitative structure of the problem at hand. There are numerous examples of exploiting problem structure in the literature, outlining a more broad stream of thought.

For example Full and Koditschek (1999), have demonstrated how one can collapse the dimensionality of a given system by trimming away degrees of freedom, exploiting synergies and symmetries. This way, one can arrive at a reduced model, the template, that can be used as a guide or a target for the control of the original system (Figure 1.2(a)). This way, the control is essentially performed on the template model and the output is then mapped back to the full system, with the additional joints and actuators.

Inspired by the *Burridge-Rizzi-Koditschek* idea of sequential switching controllers (Figure 1.2(b)), Conner et al. (2009, 2003, 2006), has shown how one can break the full state space of the system in smaller domains, each of which can be covered with an invariant control strategy, e.g. converging to an edge or a point in the domain (Figure 1.3(a)). A global model of the local domains is built from the adjacency relationships, forming a graph. Achieving a given goal would then require finding a path from start to end on the graph, coordinating the invariant local control strategies and following the flow-through policy that emerges.

Two research groups, Dever et al. (2006, 2004) and Frazzoli et al. (2005, 2003), both working on controlling *UAV* helicopters, have shown that even in difficult control problems one can organize solutions in manoeuvre sets. This way one can arrive at groups of control strategies, appropriate for each manoeuvre set, via expert demonstration or offline optimization. In turn find bounds of the manoeuvre groups, feasible manoeuvre initiation conditions and possible manoeuvre exit conditions, in essence discovering which manoeuvres can be sequenced. This is used to build a manoeuvre



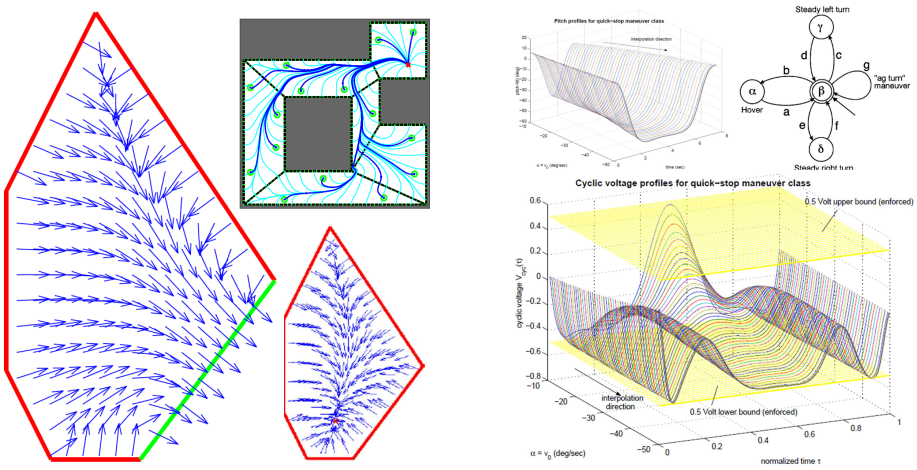
(a) Templates and anchors. (b) Seq. switching controllers.

Figure 1.2: Examples in the literature showing how one can exploit and/or impose problem structure. (*a*) The templates and anchors view of Full and Koditschek (1999), where high-dimensional neuromuscular systems can be trimmed down to reduced template models by exploiting synergies and symmetries. The template model is used for control and the calculated control actions are then mapped back to the full system. (*b*) The sequential switching controller idea from Burridge et al. (1999), where local simple controller can be used to incrementally build up a controller network covering the space of interest. Notice how the outcome of one control law brings the system to the influence of the following controller.

graph, much like a state machine (Figure 1.3(b)), and use this graph to plan paths that achieve given goals.

All the aforementioned examples still required a substantial amount of model knowledge and meticulous fine-tuning. Nonetheless, the common idea they share is that good trajectory generation approaches produce qualitatively similar solutions that are inherently structured. The central idea of this thesis is to capture the structure of such solution sets in a form appropriate for motion planning and control.

We build a framework that can concisely encode the qualitative structure of a skill, as this can bring down the computational complexity. It allows us to learn such an encoding from example solutions, as working with the full models of the systems in question is too expensive. Our framework allows for the dynamical addition of constraints either in the joint space or the task space of the system. It provides the machinery necessary for the rejection of perturbations beyond sersorimotor noise. Finally, it operates in an efficient manner, capable of reactive behaviour, permitting on-line deployment.



(a) Flow-through policies.

(b) Maneuver sets and graphs.

Figure 1.3: Examples in the literature showing how one can exploit and/or impose problem structure. (*a*) Flow-through policies, introduced in Conner et al. (2009), as in Figure 1.2(b), can be composed of smaller domains where the local invariant controllers are easier to define. Such policies can be point- or edge- converging vector fields, suitable for the plan at hand. (*b*) Dever et al. (2006) and Frazzoli et al. (2005) demonstrated how even in difficult control problems one can group control strategies, in this case helicopter manoeuvres, and arrange these into a control graph. Planning can then be performed by finding paths on this graph, picking the sequence of manoeuvres that would bring the system to the desired state.

The strategy we will use for this is to learn each skill in an *offline* phase directly from demonstrated data. Such data can be either the result of an expensive optimization procedure or expert demonstrations. Each set of demonstrated solutions belongs to a specific skill and invariantly possesses inherent structure[1]. Each skill is represented as a skill manifold, an encoding that allows for a variety of geometric operations. The set of such learnt skill manifolds can be then used by the robot *online*, in autonomous manner, accommodating novel constraints and goals. Figure 1.4 provides an overview of the proposed approach.

In general, we will see that solution sets of robotic skills have inherent structure that is closely captured with our manifold representation. We provide a number of low dimensional examples where such structure can be directly visualized. We present a method for learning skill manifolds from demonstrated solutions and an algorithm for generating novel trajectories as geodesics on such learnt geometries. We intro-

[1]More for this argument in the following section..

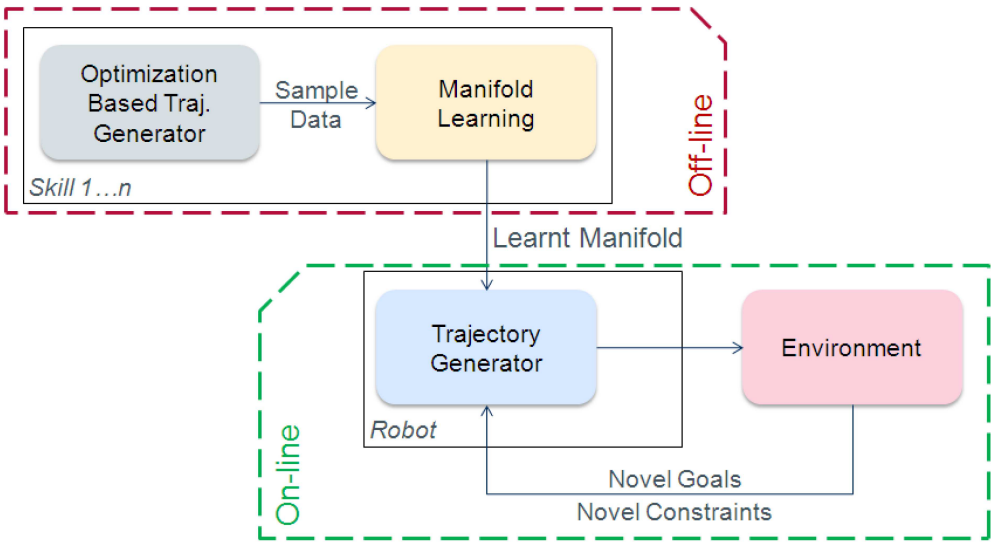


Figure 1.4: Overview of our approach. Skills are encoded as skill manifolds that are learnt from example solution sets offline. The learnt skill set is the used by the robot to generate trajectories that accommodate novel goals and/or constraints, and to react to unforeseen disturbances in an on line manner.

duce a method for adding constraints at *runtime* to such learnt manifolds, as a central requirement is that the robot should be able to operate autonomously in a dynamic environment. We demonstrate an system for online reactive control that is based on a metric naturally derived from the manifold representation. This can be used to overcome perturbations and produce strategic changes in the behaviour of the controller, stemming from the natural separation between *on-manifold* and *off-manifold* control.

Experiments are reported in order to validate the methods and to assess the performance of the various algorithms developed. In these, learning is performed on data from various optimization procedures, on systems ranging from a simple 3-link planar arm to a high-dimensional humanoid robot. Overall we illustrate that approaching skill-specific motion planning and control in terms of manifolds of example solution sets can yield significant performance benefits with regards to both behavioural stability, in the sense of a graceful degradation of the methods' generated solutions; and computational efficiency, allowing for real-time deployment.

Why model solution sets as manifolds

Robots are physically grounded systems and consist, mostly, of rotational and translational elements. Thus, the configuration spaces of most robotic systems can be nat-

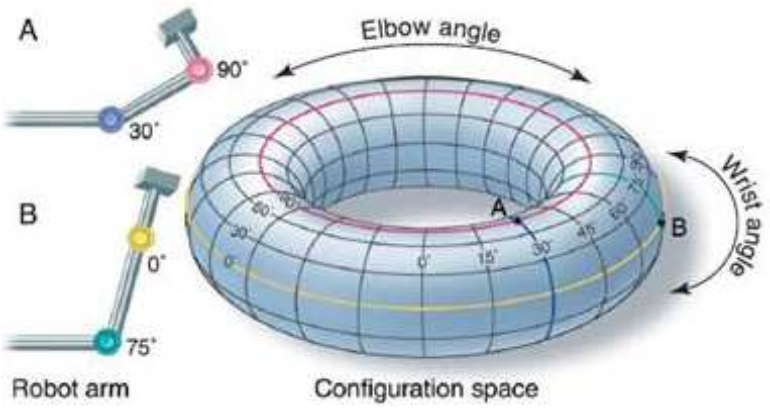


Figure 1.5: The configuration manifold of a typical two-link manipulator.

urally described in terms of products of spatial motion and rotation groups (Arnold, 1989; Novikov and Taimanov, 2006), e.g., the special orthogonal group $SO(3)$ and the special Euclidean group $SE(3)$. A pictorial example is that of the simple two-link rotational manipulator (Figure 1.5), the configuration space of which can be naturally expressed as a torus.

Definition 1 (Lie group)**.** *A Lie group is a smooth manifold obeying the group properties and that satisfies the additional condition that the group operations are differentiable.*

In general, the dynamics of most robots evolve on *Lie groups* and have the structure of both a group and a differentiable manifold (Lewis, 2007). *Lie groups* of robot dynamics are far too complex to learn from data or represent with a reasonably computationally complex model (there is some very interesting recent work with low-dimensional systems in Kobilarov and Marsden (2011)). Due to limits on the motion of actuators, the true configuration space is invariably a manifold with a boundary (Novikov and Taimanov, 2006) that defines the set of valid configurations. In turn, solutions to specific tasks impose a structure to the configuration space that is specific to the task at hand.

We know that solutions from optimal control form sets that are inherently structured, e.g., Figure 1.2, as solutions to *similar* optimal planning and control queries tend to be *similar* on a local scale. On a larger scale, such skill-specific solutions form manifolds that can be perceived as high-dimensional surfaces (*hypersurfaces*). There is an abundance of manifolds that can form in the configuration space of a complex (enough) system. It would be very difficult to arrive at a general parametrization of such a set of manifolds that accounts for a number of skills of interest in unison.

Instead we are interested in learning from data a non-parametric model of a single skill at a time. Such models are invariantly local as each hypersurface (manifold) is assumed to be locally smooth. Each such manifold models the geometric form that a set of solutions, belonging to a specific skill, trace in the space that they evolve. In principle, this space can be any arbitrary state space but throughout this work we focus on configuration spaces. By modelling each skill as a manifold of a demonstrated solution set we bring down the computational complexity of both the modelling and the generative phases, while we gradually build-up an *expressive* skill library. *Skill-graph* approaches can then be used to organize solution sets and allow for planning queries and reactive control that involve the consecutive invocation of a number of skills.

Thesis outline

In this section we provide a short outline of the thesis, highlighting the core ideas of behind the structure of each chapter. Each chapter overview is followed by a list of references to articles in which the work has been published during the course of the research, as well as an illustration of the key original contributions of each chapter.

In summary we first present our manifold learning for motion planning and control framework and the ideas that motivate our approach. Then we introduce each of the key three components of the framework and present results in stages. Last we present an overall experiment that ties in all the components of the approach, leading to a comparison on multiple levels between our framework and a state-of-the art imitation learning method.

In *Chapter 2*, we present work related to path planning, control and machine learning. The first section provides a review of methods in the field of path planning and control, while the second section presents related research in the field of machine learning with a focus on manifold learning approaches.
Original contributions:

- Review of state of the art path planning and control approaches in terms of classical, sampling based and learning based methods.
- Analysis of motivation behind using a manifold learning representation for path planning and control.
- Overview and comparison of state of the art manifold learning methods.

In *Chapter 3*, we give a detailed presentation of the manifold learning approach, central to our framework. We show how such a representation can be learnt from data and give an illustrative example on the swiss-roll dataset.
Original contributions:

- Introduction the use of a geometric manifold learning approach to robotic task representation.
- In-depth description of how the model is set-up and trained from an example solutions set.
- Analysis of the limitations of the manifold learning method as well as the benefits associated with the use of such an encoding for robotic tasks.

Publications

- Havoutis, I. and Ramamoorthy, S. (2010). Geodesic trajectory generation on learnt skill manifolds. In Proceedings of IEEE International Conference on Robotics and Automation (ICRA), 2010.

In *Chapter 4* we present a method for generating novel solutions from learnt manifold representations. We show how such solutions can be described as geodesic trajectories of the learnt manifold hypersurface and demonstrate the effectiveness of the manifold approximation in terms of computational efficiency, generalization ability and model accuracy.
Original contributions:

- Numerical optimization method of generating consistent training data as ground truth for robot learning.
- Method of generating robot trajectories as geodesics on learnt skill manifolds, minimizing path length while evolving on the underlying geometry.
- Numerous experiments presented, evaluating the algorithm's performance with respect to ground truth data.

Publications

- Havoutis, I. and Ramamoorthy, S. (2010). Geodesic trajectory generation on learnt skill manifolds. In Proceedings of IEEE International Conference on Robotics and Automation (ICRA), 2010.

In *Chapter 5* we present an extension to the trajectory generation method, that enables our framework to handle novel, time-varying or transient constraints, typical of a realistic environment. Such constraints can live in the task space or the joint space of the system and were not present in the manifold learning phase.
Original contributions:

- Extension to the unconstrained trajectory generation procedure, allowing for a dynamical addition of novel constraints.
- Introduction of an iterative procedure for generating constrained geodesic trajectories.
- Numerous experiments of unconstrained and constrained solutions to novel planning queries, in a variety of systems.

Publications

- Havoutis, I. and Ramamoorthy, S. (2010). Constrained geodesic trajectory generation on learnt skill manifolds. In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2010. (*Finalist for Best RoboCup Paper Award*)

In *Chapter 6* we propose a new approach to reactive control of humanoid robotic systems by utilizing a learnt manifold and a correspondingly derived cost hypersurface, in a model free setting. We show how we can set-up a vector field in the ambient space that enforces convergence to the desired family of trajectories from off-manifold points.
Original contributions:

- Method for reactive control on learnt skill manifolds based on an iterative procedure equivalent to the geometric projection.
- Example of a reactive control architecture based on the manifold representation.
- Analysis of the metric that arises from the manifold encoding and comparison to a naive metric, in the context of cost (value) driven planning and control approaches. Evidence of the superiority of the manifold metric.
- Numerous examples of the reactive manifold controller in scenarios where the successful execution of the task is sensitive to the cost of the solution. Demonstrations on a variety of systems.

In *Chapter 7* we present a rigorous evaluation of our manifold learning 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 retargeting, both with respect to initial and goal positions.
Original contributions:

- Numerous comparisons on a variety of metrics, as well as with regards to robustness against small and large changes of initial and goal positions.
- Analysis of the advantages and the difficulties of both methods, in the context of a realistic pick-and-place task.

Publications

- A journal paper covering reactive control and comparison with the state-of-the-art is currently in preparation.

Finally, in *Chapter 7*, we give conclusions and suggest directions for future work.

Chapter 2

Background

This chapter presents work that is related to the research goals of this thesis. It consists of two sections; the first presents work in the field of path planning, while the second presents related research from the field of machine learning and in particular manifold learning. Both fields are core to this thesis subject, as the proposed framework is adopting a manifold learning perspective for solving, in essence, path planning and control problems.

2.1 Path planning and control

The issue of path planning and control is central to almost any robotics scenario where the plant consists of physical articulated and actuated structures that are designed to interact with the physical environment.

In the general case *path planning*[1] is concerned with finding a feasible path between two defined states. Adopting a configuration space notation, as used throughout this work, path planning is concerned with computing an appropriate (smooth, feasible, collision-free, etc.) path that takes the current state of the system, q_s, to a desired (final) goal state q_g. The path that the system has to traverse in configuration space, Q, can be discretised to q^i with $i = 1,\ldots,n$, or be time indexed as $q(t)$ with $t = 1,\ldots,T$.

Many approaches to generating such paths have been developed during the past decades. For example a large body of research has been concerned with the probabilistic generation of collision free paths in an exploratory fashion. These are variations of the *Rapidly exploring Random Trees* (RRTs) detailed in subsection 2.1.5.1. Recent developments try to incorporate task constraints to RRT-based algorithms and couple the

[1] The term *motion planning* is also used in the literature.

sampling process with an analytical-model-specific corrective action (Berenson et al., 2009a, 2011). Others have looked into imitating human demonstration in generating such paths, where the goal point can be arbitrarily altered while the qualitative properties of the produced motions remain similar. Such approaches fall under the *Leaning by Demonstration* category and are more thoroughly explained in subsection 2.1.3.

The problem of control consists of generating the actions that the system must realize for following defined paths, supervising the execution of the subsequent transitions and applying corrective actions when a deviation from the path is observed or when a perturbation occurs. A common control strategy is to utilize feedback from the plant to correct with respect to a reference state. A common feedback-error based approach, very popular in robotics, is *PID* control and variants thereof, where corrective actions are taken based on a proportional, an integral, and a derivative term of an error signal.

Much of the popularity of PID can be attributed to its ease of implementation and broad generality. For example, to set-up a PID controller one needs a reference state and the current state, or estimate, of the controlled variable. This way an error term, $e(t)$, its integral and its derivative, dictate the controllers' output as:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{d}{dt} e(t), \tag{2.1}$$

where K_p, K_i, K_d are the gains of the proportional, the integral and the derivative terms of the error accordingly.

Controllers for robotic systems can be defined in many different levels. Low-level controllers operate directly on the joint level and regulate the torques applied at each joint, i.e. the force that the joint actuator should apply for the joint to reach a desired state. Such controllers are often *PD* based where the P and D gains are subject to tuning, specific for the needs of the system at hand. High-level controllers define the control actions that the system needs to execute to follow a trajectory, where it is often the case that such trajectory is indirectly defined by the controller actions. Such controllers often provide a local feedback-stabilizer, a local feedback law that would overcome any perturbations occurring during the execution of the trajectory.

In systems where analytical models are available there exist a number of control approaches that can provide analytical solutions (Stengel, 1995; Siciliano and Khatib, 2008; Levine, 1996). These methods often require solving the Hamilton-Jacobi-Bellman equation, a partial differential equation which is central to optimal control (Sontag, 1998), solved over the whole of state space. In complex systems such analytical treatments are not computationally possible. A number of approaches

have been developed for the control of complex systems, a broad variety of which we overview in the following subsections.

In general what we would like is a framework that yields a planned path and the control actions that are associated with the realization of the plan. Approaches also exist where the path is not explicitly defined, instead a vector field is imposed on the robots configuration space, based on the goal at hand, that makes the system converge to the desired configuration space point. This is a more reactive approach that draws parallel to dynamical systems theory as the configuration point of convergence can be easily considered as an attractor point in the systems phase space. Constructing such attractors has proven to be far from trivial for most interesting systems.

2.1.1 Traditional planning and control approaches

Planning is a research field central to robotics. Planning approaches can be naturally divided into task space planning, where a plan is devised for an end-effector to follow, and joint space planning, where a plan is computed for each DoF of the system to follow. Forward and inverse kinematics and dynamics can provide a mapping between task space and joint space plans but it is often the case that inverse mappings are not unique or suffer from a variety of computational difficulties.

Traditional planning approaches of low complexity make use of *splines*, a generalization of polynomials, as a structured way of creating smooth and differentiable curves for task and joint space plans (Craig, 1989; Siciliano and Khatib, 2008). This is often based on straight-line trajectories in order to limit the polynomial factors to a unique solution.

More complex approaches to planning problems include optimisation-based methods where there exists a cost function to be optimized over and in essence picks the one solution over the set of all possible solutions to a planning query. Various cost functions have been applied to movement systems. Some of the high level cost functions used are *minimum jerk* (Flash and Hogan, 1985), *minimum torque* (Uno et al., 1989) and *minimum endpoint variance* (Wolpert and Harris, 1998). Such approaches face the problem of large computational cost as the optimisation depends on complicated mathematical models. Moreover, complex movement systems in general have to utilise combinations of cost functions. Recognizing the individual cost functions that shape a particular motion is not always possible. In addition, the solution of combinations of cost functions in a mathematical model can be intractable or non-convergent.

In the most general case controllers are defined over the state space of the system. Such controllers do not depend on timing variables and provide stability guarantees on the control space. In simple systems such solutions can be computed analytically (Stengel, 1995). Textbook examples of such controllers are the Linear Quadratic Regulator (LQR) (Athans and Falb, 1966; Craig, 1989) and the Linear Quadratic Gaussian (LQG) (Stengel, 1995; Siciliano and Khatib, 2008). Much of the recent effort in this direction has focused on iterative and approximate methods of computing such controllers (Li and Todorov, 2006; Todorov and Li, 2005; Kalakrishnan et al., 2011; Theodorou E. and S., 2011), as the computational complexity of the analytical approaches quickly scales out of reasonable bounds as system complexity increases.

Control oriented approaches often assume an obstacle free workspace where no explicit path-planning phase is necessary. This is true for strict industrial-floor applications but inadequate for systems designed for close interaction with humans in everyday tasks. One of the main reasons behind this is that optimal control approaches have a difficulty factoring in a model of the system's environment. On the other extreme, often planning methods do not take into consideration the control phase of the system, potentially producing trajectories that the controller is unable to follow or clashes severely with the system's natural dynamics. One of the main reasons behind this is that optimality measures, at the control level, are hard to be factored in the planning step. Our proposed framework aims to integrate planning and control under a single representation, thus tightly couple the planning and control phases. A more detailed overview of general control and planing literature follows.

2.1.2 Vector fields

The notion of global vector fields has been a research topic for many years. A central claim of such approaches is that decoupling feedback an motion planning can be inefficient. Typically, paths computed by planners may not be smooth and can be difficult to track. On the other hand, it can be difficult to design feedback control strategies that take into account non-convex constraints induced by the obstacles in the environment. This becomes more problematic when a robot has to replan as a consequence of a feedback controller failing to steer the robot to follow a prescribed path.

Global vector field approaches, rather than planning a single path from start to goal, try compute a feedback law, in the form of a vector field, over the entire free space of the system. Early vector field approaches Choset et al. (2005) would naively

set up sources and sinks, on constraints and targets respectively, with the use of simple structures such as *Radial Basis Functions* and then follow the resulting vector field to arrive at the global steady state (sink).

This of course tends to be problematic when the free space is complex, rather typical of multi-DoF configuration spaces. The most common issue is that the resulting vector field can contain spurious attractors that can trap the system state in local minima. More recent work on global vector fields moves against this problem by assembling a global solution through local decompositions. For example, Zhang et al. (2009) preform an approximate cell decomposition of the free space and compute vector fields within cells by considering their adjacency relation, with the constraints and goals in mind. Another clever approach from Conner (2008) takes another path by, again, decomposing the full space in smaller domains and setting up local *predefined* feedback laws with specific state-evolution properties.

All such approaches have been demonstrated to work well with low-dimensional configuration spaces and systems that can easily be locally linearly approximated. These approaches scale badly with increasing complexity for reasons of decomposition and neighbourhood calculation. Cartesian space planning is nicely suited by global vector field approaches but often planning and feedback in configuration spaces yield poor results.

2.1.3 Learning by demonstration

Learning by Demonstration (LbD), sometimes also in the literature as Programming by Demonstration (PbD), has appeared as one way to respond to the growing need for intuitive control methods (Billard et al. (2008)). LbD belongs to a larger class of approaches of *Imitation Learning*, where the general goal of which is to have the system at hand learn from examples (Argall et al. (2009); Billard et al. (2007); Schaal et al. (2003)).

In LbD the user provides a small set of demonstrations, usually 5 to 10 examples, and the system should be able to encode the dynamics of this motion for future use. Such representation is time independent and at its core is the ability to locally estimate the dynamics of the example-generating system ,f.

Estimating the dynamics of f given a set of demonstrations is done with a statistical approach; *Gaussian Mixture Regression* (GMR). This approximates f with and estimate $\hat{f}$ as a nonlinear combination of a finite set of Gaussian kernels that define

a joint probability distribution function (PDF), $P(\xi^i, \dot{\xi}^i)$, over a training set of example trajectories $\xi^i, \dot{\xi}^i, i = 1, \ldots, M$. The PDF is defined as a mixture of K Gaussian $G^1, \ldots, G^K$, with μ^K and Σ^K being the mean and covariance matrix of a Gaussian G^K:

$$P(\xi^i, \dot{\xi}^i) = \frac{1}{K} \sum_{k=1}^{K} G^K(\xi^i, \dot{\xi}^i; \mu^k, \Sigma^k), \tag{2.2}$$

and

$$\mu^k = [\mu^k_\xi; \mu^k_{\dot{\xi}}] \text{ and } \Sigma^k = \begin{pmatrix} \Sigma^k_\xi & \Sigma^k_{\xi\dot{\xi}} \\ \Sigma^k_{\dot{\xi}\xi} & \Sigma^k_{\dot{\xi}} \end{pmatrix}, \tag{2.3}$$

where each Gaussian probability distribution G^k is given by:

$$G^K(\xi^i, \dot{\xi}^i; \mu^k, \Sigma^k) = \frac{1}{\sqrt{(2\pi)^{2d}|\Sigma^k|}} e^{-\frac{1}{2}(([\xi^i_t, \dot{\xi}^i_t] - \mu^k) T (\Sigma^k)^{-1} ([\xi^i_t, \dot{\xi}^i_t] - \mu^k))}. \tag{2.4}$$

The model is trained with the *Expectation-Maximization* algorithm (EM) and generation of a new trajectory from a learned *Gaussian Mixture Model* (GMM) is done by sampling equation 2.2, this process is termed GMR (for further reading on GMR the interested reader can consult Calinon and Billard (2008)).

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. A difficulty common to almost all imitation learning approaches is that of scaling up to a system with multiple DoFs and complex dynamics. 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.

2.1.4 Dynamic motion primitives

Another approach that is directly comparable with LbD is that of *Dynamic Motion Primitives* (DMPs). DMPs provide a framework for learning the dynamics of demonstrated motions by using a simple dynamical system 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.

DMPs can be used to generate discrete and rhythmic movements. Each DoF is represented by a coupled dynamical system of the form:

$$\tau\dot{u} = K(g-x) - Du + (g-x_0)f, \tag{2.5}$$

$$\tau\dot{x} = u \tag{2.6}$$

where x and u are positions and velocity fo the system, x_0 and g are start and goal position, τ is a temporal scaling factor, K acts as a spring constant and D as a damping factor. The factor f is the non-linear function which allows the generation of complex motions and is the part that is learnt from data. The nonlinear function is of the form:

$$f(s) = \frac{\Sigma_i w_i \psi_i(s) s}{\Sigma_i \psi_i(s)}, \tag{2.7}$$

where $\psi_i = exp(-h_i(s-c_i)^2)$ are Gaussian basis functions with center and width, c_i and h_i accordingly and w_i adjustable weights. This function is also coupled with a phase variable s which moves from 1 to 0, signifying that the function's effect on the output progressively dies out and the canonical dynamics of the attractor system eventually prevail. This timing variable evolves as:

$$\tau\dot{s} = -\alpha s, \tag{2.8}$$

where α is a predefined constant. The set of the above equations define one DMP and a set of n DMPs would be needed to represent a motion in an n-dimensional state space.

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. In other words, the produced movements are self-similar for changes in start positions, goal positions and temporal scaling. In addition the 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.

Nonetheless, such representation requires the tuning of many parameters and is limited to encoding one DoF per DMP, something that limits its capability of capturing the interplay between the DoFs of a controlled system. Furthermore, learning is performed on a single demonstration, thus limiting the generalization ability of the approach.

The DMP framework was introduced around a decade ago by Ijspeert et al. (2002) and has been further developed and extended over the years by a couple of groups since then (Schaal, 2006; Degallier et al., 2006; Ijspeert et al., 2003; Nakanishi, 2004).

Recently Pastor et al. (2009) have shown how to alleviate the effect of unwanted, often excessive, scaling that resulted from changing the goal of a system. Here as well there has been a transition from joint space to task space, which of course make the variables in question much more "*well-behaved*", and an additional (given) inverse kinematics layer is used to produce the actual joint movement of the plant.

2.1.5 Sampling based motion planning

Planners based on *road-maps* (RMs) have been used in early robotic path planning problems, mostly involving planar mobile robots and low degrees-of-freedom (DoFs) manipulators. The main idea was to divide the configuration space of the plan in free and obstacle space and form a graph-based approximation for traversing the free space. With the increase of the platforms' DoFs, having such an explicit representation of the configuration space started to become infeasible. This, in turn, led to the formulation of Probabilistic RMs (PRMs), where the free space was explored in a probabilistic fashion and the RM was built out of the constraint-respecting samples.

As plant complexity increased and environments became more dynamic, simple sampling approaches had to be reconsidered. Specifically, there was a need for 'single query' planners operating in an on-line manner. This lead to Rapidly-exploring Random Trees (RRTs). The simple idea behind RRTs is that one does not need to exhaustively sample the configuration space entirely before looking for a single path.

Instead one can sample randomly in a structured manner, which ensures connectivity between the samples - thus at the same time building up the connectivity graph in an RM-like fashion. In addition building on an RM, storing and managing such an amount of information quickly becomes infeasible as the plant complexity increases. RRTs are single query algorithms, thus after a solution is found the resulting tree is discarded as there is no guarantee that it would equally be valid under a different planning query. Moreover the use of structured sampling ensures uniform coverage of the space that is explored. The algorithmic machinery behind RRTs is quite simple and straightforward - an important factor in spreading the use of RRTs in a wide variety of applications.

2.1.5.1 The RRT algorithm

RRT (LaValle, 2006) is a remarkably simple yet effective algorithm for planning a path between two points in configuration space. All the algorithm needs for operation are

start and end points, as well as a distance metric for evaluating sample points.

In the algorithm, one adopts a simple set characterisation of the configuration space, which is the union of the free space, Q_{free} and the obstacle space Q_{obs}. In turn all samples q belong to Q_{full}:

$$\forall q \in Q \Leftrightarrow Q_{full} = Q_{free} \cup Q_{obs}.$$

Q_{full} can be the configuration space or the phase space for the system, or even just any composition of state variables within $q \in \mathbb{R}^D$, D being the dimensionality of the problem space.

We root a tree, T, at the given starting point, q_{init} and grow it by iterating the following process. Pick a random point $q_{rand} \in Q_{full}$ and calculate its distance from each point already in T. Select the closest point from T, q_{near}, and grow the tree toward q_{rand} by a step size Δx. Then evaluate if the resulting configuration

$$q_{new} = q_{near} + \Delta x^{q_{rand}},$$

belongs to Q_{free} or Q_{obs}. If the former is true q_{new} is added to T, alternatively the sample is discarded. The procedure is repeated until the goal configuration q_{goal} is reached, within some tolerance or number of iterations. The shortest path is then computed on T using a tree search algorithm. Pseudocode of the classic version of RRT is presented in Algorithm 1 while a schematic representation of the algorithm's steps is available in Figure 2.1..

A nice feature of the RRT is that there are few tunable parameters that need to be experimented with. Furthermore, RRTs are probabilistically complete. Thus, given that a solution exists RRTs are guaranteed to find it with increasing probability as the number of samples grows.

In addition RRTs do not require an explicit model of the system that is being sampled. It suffices to have a way of evaluating new samples according to the set of planning constraints that might be relevant to the problem at hand. For example when considering a reaching task the evaluation might involve collision detection and when considering a walking task the evaluation might be more relevant to the stability of the plant. RRT planners are thus much more flexible in the sense that they can work with any simulated or real model that can be treated as a black-box system. Nevertheless an accurate model of the plant's dynamics and kinematics can contribute to the planning speed of the algorithm as the evaluation of the samples can be derived numerically.

Algorithm 1 Rapidly-exploring Random Tree

RRT(q_{init}, q_{goal})
INPUT: start point q_{init}, goal point q_{goal}
OUTPUT: path in configuration space p
T.add(q_{init}) {Initialize tree T}
for $i = 0$ to k **do**
  $q_{rand} \leftarrow$ RANDOM_POINT
  $q_{near} \leftarrow$ NN(q_{rand}, T)
  $q_{new} \leftarrow$ STEP(q_{near}, q_{rand}, dx)
  $valid \leftarrow$ EVALUATE(q_{new})
  if $valid == true$ **then**
    T.add(q_{new})
    $dist \leftarrow$ DISTANCE(q_{new}, q_{goal})
    if $dist \leq tolerance$ **then**
      break
    end if
  end if
end for
$p \leftarrow$ SHORTEST_PATH(T.first,T.last)
RETURN p

However, when considering complex problems involving humanoids, many finer points need consideration, including convergence to the goal, stability and realisability constrains, space coverage and resolution. For example, in spaces with $D \geq 8$ convergence is typically slow. It has been shown that including a small bias, e.g. 0.01 favouring the goal, greatly increases the convergence speed as it steers the exploration 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 issue 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 other approaches.

The past few years have seen some very exciting developments in the area of motion planning for humanoid robots and other complex architectures. Kuffner et al.

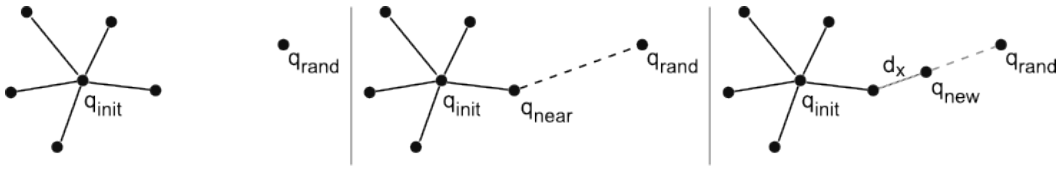


Figure 2.1: Schematic representation of the RRT algorithm. The search tree is rooted at q_{init} and at each iteration a random configuration point, q_{rand}, is picked. The nearest point that belongs to the search three is computed (q_{near}) and a step, dx, is taken towards the random point. If the new point results in a valid configuration, with regard to the planner's constrains, it is added to the search tree and the procedure is repeated until the goal is reached.

(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.

In general, overall success of such algorithms depends on the metric that is defined over the space to be explored. Traditionally a metric of the form:

$$d(q,q') = \sum_{i=1}^{n} w_i \left\| q_i - q_i' \right\|,$$

is used where the weights w_i denote the importance of each Degree of Freedom (DoF). These weights are often empirically chosen based on trial and error. As the dimensionality grows in nonlinear systems, this becomes difficult as our intuitive notions of neighbourhood become inapplicable. These are the sorts of spaces where modern humanoid robots live. There is a need for other ways to derive such metrics. We argue that learning such a metric in a data-driven fashion is a desirable and scalable approach.

The second, related, issue that determines success of RRT-based planning is coverage. Random sampling in high dimensions can be excessively wasteful when the underlying task has very special structure. A bulk of recent work has demonstrated that due to a variety of reasons, e.g., joint limits, self-collisions, stability and energy constraints, 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). It is desirable to leverage this in the process of achieving better sampling coverage as the dimensionality of the problems increases.

If these manifolds have non-zero volume in the C-space it is straightforward to show that an RRT-based algorithm is probabilistically complete as in the limit of sam-

ples, sampling will eventually place samples inside the manifold. However, if such a manifold has zero volume in the C-space, a hypersurface in the high dimensional space, then the probability of sampling such a subspace is practically zero. An approach that uses a projection operator to move samples from the high dimensional space to the surface has been used in Berenson and Srinivasa (2010). We have explored a similar idea in Havoutis and Ramamoorthy (2010c) with limited success.

2.2 Machine learning

Systems of interest that interact with the environment in a physical manner are hard to accurately model. In recent decades there has been strong turn towards learning paradigms.

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. Wang et al. (2008) introduced GPDM, a Gaussian processes based dimensionality reduction with a dynamical model of the evolution of the state, that can learn models of human kinematic trajectories. In the same spirit, 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 parametrised path generation method over the learnt representation to generate novel movements.

2.2.1 Motivation to use a manifold learning approach

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., 2009b; 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.

An approach that is closer to our current work is that of Calinon and Billard (2009), who demonstrate robot programming by demonstration with a probabilistic model, namely Gaussian Mixture Regression, of Jacobian-based inverse kinematics for learning trajectories and incorporating task space constraints. What has not always been exploited in such work is the *geometrical structure* of families of paths in lower dimensional subspaces. By simply mapping a set of poses to a low-dimensional space and fitting a parametrised model, one is essentially *overriding* potential intrinsic dynamics effects that define many behaviours of interest.

Our goal is to learn this 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*.

2.2.2 The manifold representation advantage

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. Popular 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, we are interested in preserving properties of trajectories in the data set. So, our goal is to learn a model of the tangent space of the low-dimensional nonlinear manifold, conditioned on the 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.

An important point that differentiates our approach from alternate data-driven approaches that also utilize some form of dimensionality reduction (Vijayakumar et al., 2005; Bitzer et al., 2008; Full and Koditschek, 1999; Ramamoorthy and Kuipers, 2006;

Jenkins and Mataric, 2004) 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, 2010a) 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.

2.2.3 Manifold notation and concepts

A *manifold* is a space where every point has a neighbourhood homeomorphic to an open Euclidean n-ball, an n-dimensional space following an Euclidean distance metric, where n is allowed to vary. Manifolds can consist of a single connected component or be disjoint unions of connected components. Manifolds that have a fixed n (i.e. n-dimensional manifolds) are called *pure* manifolds. For example, the sphere has a constant dimension of 2 and is therefore a pure manifold whereas the disjoint union of a sphere and a line in three-dimensional space is not a pure manifold as the dimensionality of the line is 1.

Definition 2 (Manifold)**.** *A manifold is a topological space that is locally Euclidean, i.e., around every point, there is a neighbourhood that is topologically the same as the open unit ball in* $\mathbb{R}^n$.

Generally, manifolds are taken to have a fixed dimension (the space must be locally homeomorphic to a fixed n-ball), and such a space is called an *n-manifold*. Intuitive examples are; the 1-manifold that is a curve and the 2-manifold that is a surface, while higher dimensionality is not easily accessible to human intuition. 3-manifolds and above can be loosely termed as *hypersurfaces*.

In robotics, a motion manifold is usually the subspace on which a trajectory is restricted to lie on, i.e., a geodesic curve on the manifold. In this context, the manifold is a hypersurface that is embedded in the high dimensional state space of the system. This ambient space is often described as the configuration space of the system but, without loss of generality, it can also allow for higher order terms as velocities and accelerations, in this case the manifold is embedded in a *mixed* space.

Definition 3 (Geodesic)**.** *A geodesic is a locally length-minimizing curve.*

A geodesic can be considered as a generalization of the notion of a straight line in a curved space. Given a metric, a geodesic curve is defined as the a local length-optimal path between two points. In the plane, the geodesics are straight lines, while on the sphere, the geodesics are great circles. In general, geodesic curves connecting two points on a manifold are curves, that belong to the manifold (hypersurface) and are optimized with regards to their length.

A number of machine learning methods have been developed to deal with points that yield to a manifold representation. Many of such methods are concerned with finding an embedding of lower dimensionality in order to unroll, and often better visualize or classify, high dimensional data. This is often coupled with a metric preservation process that seeks to preserve the metric relations in the low dimensional embedding. Work in *Self-Organising Maps* (SOMs) has also been motivated by similar ideas though most research was focused on learning forward and inverse kinematics (Ritter, 1997; Walter and Ritter, 1995), where the SOMs were viewed as an intermediate representation between joint and task space. Initially SOMs have been considered but were discarded as an alternative because of their sensitivity to initialization in high dimensional data (Kohonen, 1997). The following subsections present some of the most significant manifold learning approaches in limited detail. The interested reader can refer to the references provided for further details, as an in-depth presentation of all these methods is omitted in the interest of space.

2.2.3.1 Coordinated factor analysis

Coordinated factor analysis (CFA) has been introduced in (Verbeek, 2006) for modelling data sampled from manifolds. It uses a mixture of factor analysers to approximate a non-linear factor manifold. The method fits local factor analysis models and then tries to coordinate the local factors in order to recover a global parametrization. The global alignment of the local models is achieved by modifying their objective function. This method has been developed in an in image analysis context and is well suited for geometrical image transformations. The model optimization is performed with EM and is prone to local minima.

2.2.3.2 Locally linear coordination

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. Given an already trained mixture, the alignment is achieved by applying an eigensolver to a matrix constructed by the internal representations of the mixture components. Research results show visualization and interpolation on high dimensional data.

2.2.3.3 Manifold charting

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. However, charting is based on a cost function much closer in spirit to the original global coordination model and it instantiates one local model centred on each training point, so its scaling is the same as that of LLE and Isomap.

2.2.3.4 Isomap

Isomap, proposed by Tenenbaum (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 that the underlying structure is a manifold with a boundary. 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. A single connected component covering this manifold approximates all-pairs geodesic distances on the underlying manifold. By applying MDS to a matrix of geodesic distances, nonlinearities in the data due to the manifold are removed to produce a coordinate space intrinsic to the underlying manifold. No explicit model is used to measure pairwise distances. Isomap relies only on measuring distances between proximal points and uses shortest-paths coordination for distances between distal points. The utilization of coordination for distal points typically provides better uncovering of structure, given sufficient data density.

2.2.3.5 Locally smooth manifold learning

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, which is crucial for use in a robotics learning context as in this disser-

tation. Learn manifolds are trained on an error function that estimates the centred directional derivative between neighbouring points in the high dimensional data set. The function is approximated with an RBF network, with the number of kernels and dimensionality being open parameters. The learnt manifold can be used to compute geodesic distances, to find projections of points on the manifold and to generate novel sample points.

2.2.4 Comparison of above methods

Common to all aforementioned methods is the idea that point sets arising from diverse problems often contain intrinsic structure. Such structure is seldom addressed in most dimensionality reduction methods. Manifold learning is a recent trend that seeks to leverage the benefits of such structure and exploit it for use in interesting problems.

Many of the aforementioned manifold learning methods are mostly concerned with embedding the high dimensional data to a low dimensional space that makes them more easily interpretable, for classification or even visual investigation. In contrast our framework focuses on an efficient representation of the demonstrated data with goal-oriented trajectory generation in mind. Many of the statistical methods presented earlier can sample the learnt models but explicitly defining a goal for the sampling procedure might be difficult.

In many of the methods discussed earlier there is no generative process. This is an essential requirement in order to create novel data, e.g., generate motions similar to the example solutions. Such generation is often done ad-hoc or in an interpolative manner which completely oversees the geometric information inherent to a skill manifold. In addition, even though the manifold structure is taken into account in the calculation of the embedding, it does not actively take part into the generative process, which often is crucial for the success and applicability of the method.

Having a low-dimensional embedding is often useful for higher-level motion recognition and classification tasks but it often disregards geometric information that can be very useful to lower-level tasks as trajectory generation and control. Our framework is chiefly aimed at motion planning, re-planning and control, thus a manifold learning method that yields an efficient geometric representation of robotic skills is paramount.

The last method that we have briefly introduced, LSML in subsubsection 2.2.3.5, forms the basis for the manifold learning part of this thesis. It combines both a geometric driven approach to learning and a generative process that utilizes the learnt ge-

ometry. In addition it is much more geared towards generalization to unseen portions of the manifold. Instead of finding an embedding for visualization, it learns a representation on the manifold. This way we gain access to a variety of geometric operations, such as computing geodesics, manifold projection, etc., that are highly beneficial in a motion planning and control context. A detailed examination of the manifold learning algorithm along with the computational model and its parameters follows in the next chapter.

Chapter 3

Manifold learning

In this chapter we will present the manifold learning algorithm that has been used throughout this dissertation. We will first present some simple manifold notions and how non linear dimensionality reduction can be framed as a manifold learning problem. We will then present the main ideas behind this method and detail the model that has been used. Learning (training) the model based on data is then explained, followed by a section that outlines the specific benefits of using such an approach.

3.1 Overview of approach

Data sets that fall on or near a low dimensional manifold have been the subject of study by a number of techniques. Linear manifolds with few dimensions are relatively easy to estimate with a small amount of data. Data that are drawn from high dimensional nonlinear manifolds are an challenging task for machine learning research. Such data sets result whenever the modes of variability of the data are much fewer than the dimension of the data space, as is the case of robot movements.

Manifold learning refers to the problem of recovering the structure of a manifold from a set of unordered sample points. Often manifold learning is equated with dimensionality reduction, especially in cases where the goal is to find an embedding or "unroll" a manifold into a lower dimensional space. Embedding is done in a way that certain relationships between points are preserved, while the dimension is 2 or 3 as embeddings are typically used for visualization.

In our framework we seek to recover the structure of a manifold by examining the motion of a point on the manifold. Our approach, based on LSML introduced in Dollár et al. (2006), attempts to learn an approximation of the tangent space that is

locally common to neighbouring data points. This way we can start from a point in the high dimensional data space and generate its neighbours, 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 that our goal is to learn how to traverse such structure rather than embed it to a new coordinate frame.

3.2 Method mechanics

Assume a given data set of D dimensions, in our case a set of example trajectories in a robot state space as discretised datapoints. Due the kinematic and dynamic constraints of the plant and the task, such 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 neighbouring points on the manifold, x^i and x^j[1] , the difference between these points, $\Delta^i_{\cdot j}$, 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 $\Delta^i_{\cdot j}$ to be the centred estimate of the directional derivative at $\bar{x}^{ij}$ and ε^{ij} to be the unknown alignment factor, we have $\mathcal{H}(\bar{x}^{ij})\varepsilon^{ij} \approx \Delta^i_{\cdot j}$, that holds given ε is small enough and the manifold can be locally approximated with a quadratic form.

Definition 4 (Tangent space)**.** *Let x be a point in an d-dimensional manifold $\mathcal{M}$, and attach at x a copy of $\mathbb{R}^d$ tangential to $\mathcal{M}$. The resulting structure is called the tangent space of $\mathcal{M}$ at x and is denoted $T_x\mathcal{M}$. If γ is a smooth curve passing through x, then the derivative of γ at x is a vector in $T_x\mathcal{M}$.*

[1] Where superscript i and j are used for indexing.

3.3 The manifold model

To model $\mathcal{H}$ assume a model that we need to optimize with respect to the given data set and this model has a set a set of parameters, here represented as θ, thus our model is $\mathcal{H}_\theta$. The error that we need to minimize is:

$$err(\theta) = \min_{\{\varepsilon^{ij}\}} \sum_{i,j \in N^i} \left\| \mathcal{H}_\theta(\bar{x}^{ij})\varepsilon^{ij} - \Delta^i_{.j} \right\|_2^2. \tag{3.1}$$

The goal of training would be to find the θ that minimizes Equation 3.1, 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 model itself. The intuition behind the first term is that the learned tangents at two neighbouring 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 $\left\|\varepsilon^{ij}\right\|_2^2$ is also constrained. Regularization is performed with the addition of the following term to Equation 3.1:

$$r = \lambda_E \sum \left\|\varepsilon^{ij}\right\|_2^2 + \lambda_\theta \sum \left\| \mathcal{H}_\theta(\bar{x}^{ij}) - \mathcal{H}_\theta(\bar{x}^{ij'}) \right\|_F^2. \tag{3.2}$$

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 $\lambda_E = \lambda_\theta = \lambda$.

3.4 Learning the manifold

In principle any regression technique can be employed for modelling $\mathcal{H}_\theta$, 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 used are of the form

$$f^j(x) = exp(\frac{-||x-\mu_j||_2^2}{2\sigma_j^2}),$$

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 neighbour center (*See Figure 3.1 for a brief overview of*

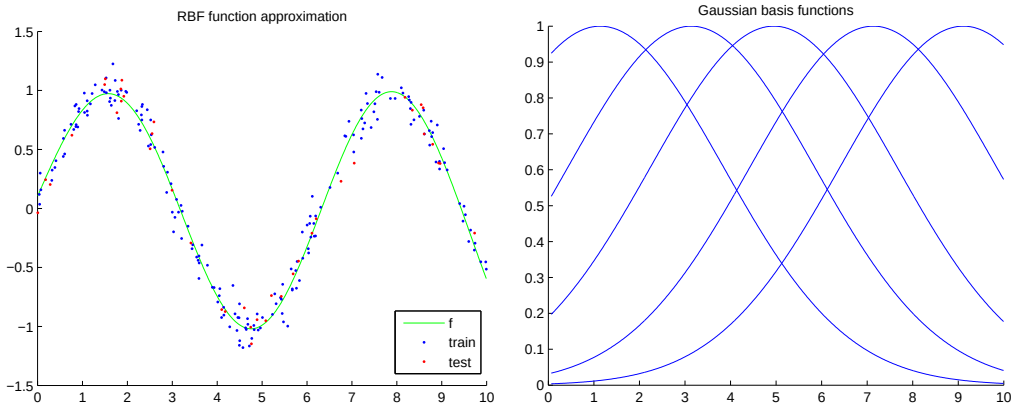


Figure 3.1: A one-dimensional sinusoid approximated by RBFs. On the left the function is plotted with green and from this function 200 noisy datapoints are sampled and used for training. The noise in this example has a variance of $\sigma = 0.1$ and the samples are plotted in blue. A set of 40 random test points are plotted in red. The left figure shows how the basis functions are spaced. The centers μ and the widths σ are computed from the training dataset, while the number of basis used k is set to 5 for this example.

function approximation with RBFs). 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 to 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 and 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 $\Delta^i = U^i \Sigma^i V^{iT}$ and by plugging in this and $\mathcal{H}_\theta(\bar{x}^{ij})$ into Equation 3.1 with rearranging we get

$$err(\theta) \quad = \quad \min_{E^i} \sum_{i=1}^{n} \sum_{k=1}^{D} \left\| \mathbf{f}^{iT} \Theta^{kT} E^i - U^i_{k.} \Sigma^i \right\|_2^2. \tag{3.3}$$

To solve Equation 3.3 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 least squares problem. The system is solved for E^i keeping the Θ^ks fixed as

$$E^i = arg\min_{E^i} \sum_{k=1}^{D} \left\| \mathbf{f}^{i^T} \Theta^{k^T} E^i - U^i_{k.} \Sigma^i \right\|_2^2 \tag{3.4}$$

$$= arg\min_{E^i} \left\| H^i E^i - U^i_{k.} \Sigma^i \right\|_F^2 \tag{3.5}$$

$$= H^{i^+} U^i \Sigma^i. \tag{3.6}$$

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

$$\Theta^k = arg\min_{\Theta^k} \sum_{i=1}^{n} \left\| \mathbf{f}^{i^T} \Theta^{k^T} E^i - U^i_{k.} \Sigma^i \right\|_2^2 \tag{3.7}$$

$$= arg\min_{\Theta^k} \left\| \left(E^{i^T} \otimes \mathbf{f}^{i^T} \right) vec\left(\Theta^{k^T} \right) - vec\left(U^i_{k.} \Sigma^i \right) \right\|_F^2. \tag{3.8}$$

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

$$vec\left(\Theta^k\right) = \begin{bmatrix} E^{1^T} \otimes \mathbf{f}^{1^T} \\ \vdots \\ E^{n^T} \otimes \mathbf{f}^{n^T} \end{bmatrix}^+ \begin{bmatrix} \Sigma^1 {U^1_{k.}}^T \\ \vdots \\ \Sigma^n {U^n_{k.}}^T \end{bmatrix} \tag{3.9}$$

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.

3.4.1 An example

The best way to demonstrate the the manifold learning algorithm is with a working example. Here we present a manifold learning task step-by-step. A widely known manifold learning benchmark is the example of the swiss-roll in 3 dimensions. The dataset in this example is randomly sampled from a well defined subspace of the 3 dimensional ambient space. This subspace has the form of a volumeless surface that is rolled into a spiral pattern (Figure 3.2(*left*)) . It is easy to see that this subspace is inherently two dimensional, thus a two dimensional embedding would suffice to describe it. The key observation in this case is that distances between neighbouring datapoints hold locally but break down for points that belong to different rolls.

Consider a dataset $\mathbf{x} = x_1, \ldots, x_n$, sampled from a swiss-roll manifold as in Figure 3.2 (*right*). The data in this case as in all but the toy examples present in this dissertation are sampled from paths that follow the surface geometry. This is typical

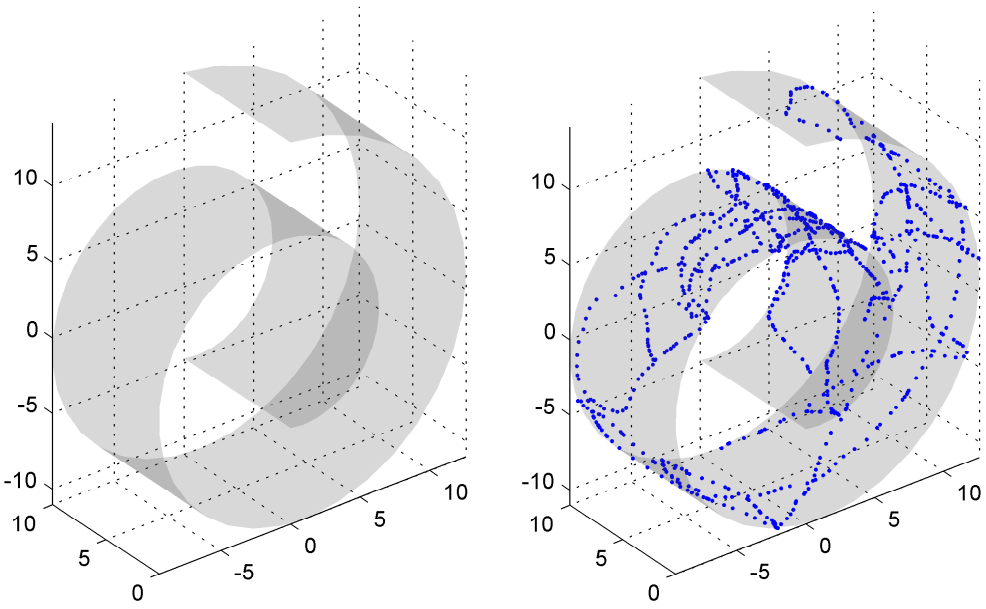


Figure 3.2: The *swiss-roll* manifold. The data comes from paths that are sampled from a surface, in a sequential and directional manner, that is embedded in the 3-dimensional space. This surface has no volume and can be unrolled to a 2-dimensional space.

of state space trajectories that follow task specific and plant constraints. First we begin by computing the nearest neighbour graph by iteratively increasing the neighbourhood distance until we have a single component, as in Figure 3.3 (*left*). The distance metric in this scenario should include all the neighbouring data points but not relate data-points that are on different rolls of the surface. Based on the neighbourhood relations we compute Δ^i that gives us an estimate of the local tangent space of the manifold at each datapoint, Figure 3.3 (*right*). Based on this the model error will be subsequently calculated for each iteration of the model error minimization loop. We set the dimensionality of the manifold to 2 as in this case it is easily apparent. This of course should not be the case in more complex problems. The correct way to set d for higher dimensional problems would be through a cross-validation procedure. Next we need to set the number of basis functions that the model will use. In this case a network of 20 RBFs was empirically found to be adequate. Again setting k can also be included in a cross-validation procedure, while one should keep in mind that this parameter can also affect the smoothness of the model.

Once all d and k have been set then we preform k-means clustering on the dataset to compute the centres for the RBFs, μ. The width of the basis functions, σ, is set according to the distance of the centres. After pre computing the features and the SVD

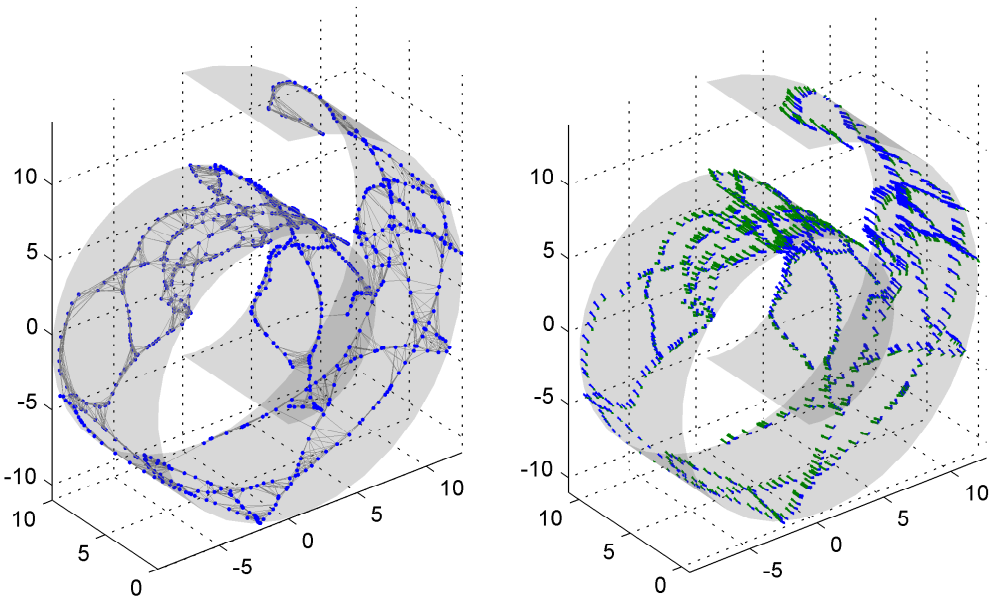


Figure 3.3: The nearest neighbor relations between datapoints. The metric holds for local neighbourhoods as relating points that lie on different rolls of the surface would result in a skewed mapping. The local tangent basis that are estimated from the dataset are plotted on the right subfigure.

of Δ^i, we initialize Θ's randomly and initiate the alternating minimization procedure. We iterate until convergence. As a result we have a model of the tangent space that is lower in dimensionality, it is compactly represented and covers the entire ambient space. Figure 3.4 presents the manifold tangent space basis evaluated at the centres of the basis functions. We can now evaluate the learnt model at every point and also create novel datapoints that satisfy the manifold geometry. This way we have a systematic way of distinguishing between points that belong to the modelled surface and points that belong to the ambient space.

3.5 Benefits of the manifold representation

Apart from the desirable properties of the manifold learning method, e.g. good generalization ability and computational efficiency, a key advantage over other methods is that it learns a geometric representation of the underlying manifold. This gives us access to a variety of geometric operations that are desirable in a robotics motion planning and control context. As we will see in the following chapters, with such a representation we can create novel trajectories from a manifold that is learnt from a set of example

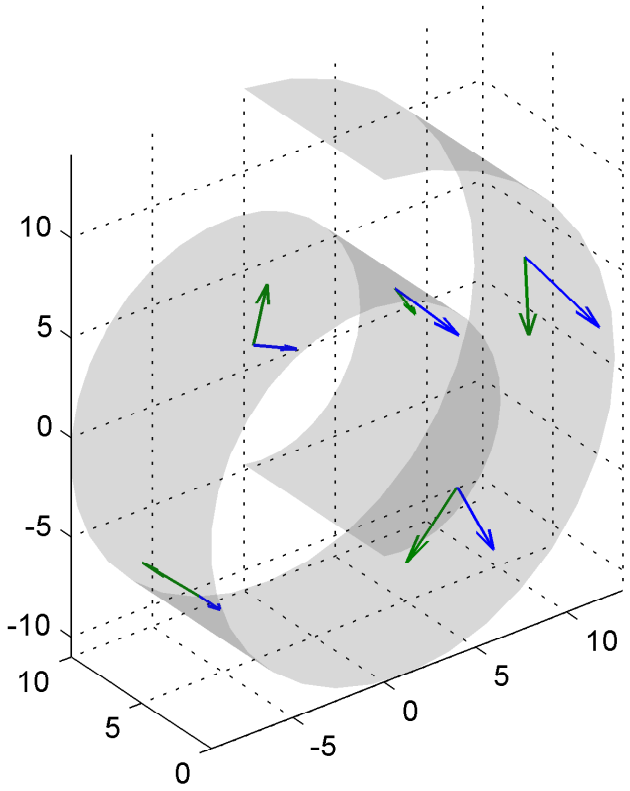


Figure 3.4: The manifold surface with samples of the learnt tangent space approximation. The representation provides a smooth mapping on the entire ambient space. Here we plot the tangent basis evaluated at the centres of the RBFs used to learn the manifold model.

trajectories. We can optimize over the model to incorporate time-varying and transient constraints that are not present at learning, and compute distances to desirable spate space locations. We can ensure the execution of a manifold following state path and find the closest return-to-manifold trajectory.

In addition a representation based on the manifold learning method that we have presented above provides a direct benefit for motion planning and control. The encoding is grounded in the system's ambient space, thus firstly it can express a global strategy for the system's motion and control, and secondly it does not require an intermediate mapping of the state for, again, controlling or planning. Such mappings are common amongst other machine learning approaches and hinder the flexibility of the system as the control in effect happens in a different space. Such benefits are elaborated in the following chapters, where the efficacy and applicability of the proposed framework is exemplified.

Chapter 4

Generation of trajectories on learnt manifolds

In this chapter we present the trajectory generation part of our approach. We first present how skill manifolds arise in robot learning. We then detail how path generation is preformed over learned motion manifolds. We will present the details of the algorithm as well as the benefits that such a learning approach can leverage. Examples and experimental results from a variety of platforms are presented.

4.1 Introduction

Humanoid robots are appealing due to their inherent dexterity. However, these potential benefits may only be realized if the corresponding motion synthesis procedure is suitably flexible. This chapter presents a flexible trajectory generation algorithm that utilizes a geometric representation of humanoid skills (e.g., walking) - in the form of skill manifolds.

These manifolds are learnt from demonstration data that may be obtained from off-line optimization algorithms or a human expert. We demonstrate that this model may be used to produce approximately optimal motion plans as geodesics over the manifold and that this allows us to effectively generalize from a limited training set.

We demonstrate the effectiveness of our approach on a simulated 3-link planar arm, and then the more challenging example of a physical 19-DoF humanoid robot and on the 23-DoF Nao humanoid. We show that our algorithm produces a close approximation of the much more computationally intensive optimization procedure used to generate the data. This allows us to present experimental results for fast motion

planning on a realistic – variable step length, width and height – walking task on a humanoid robot.

4.2 Motivation for a manifold representation

In recent years, humanoid robot platforms have been receiving increasing attention due to their inherent dexterity and great flexibility. Correspondingly, this highlights the need for general purpose motion planners. Off the shelf solutions for humanoid robot behaviours are often restricted to a limited motion vocabulary that does not exploit the full capacity of the system. For instance, predesigned motions in many platforms are not parametrized in a flexible way (e.g., allowing full control over step length, width and height) and impose a limited discretisation on the reachable space of the robot. There is a pressing need for efficient algorithms that can overcome these limitations and achieve a relatively rich set of within-skill variations in a realistic and practically implementable setting. Given such algorithms, one could then treat the skill as a component in a higher level discrete search (Chestnutt et al., 2005). Standard approaches that do allow for such flexibility tend to be computationally expensive, e.g., requiring high dimensional numerical optimization or c-space search. We need a more efficient alternative.

In realistic domains, e.g. RoboCup, where restrictions to variations on a skill would adversely impact higher level planning goals, one seeks a compact representation of the family of possible motions of a particular skill. This means that one would like to be able to learn and compactly represent the whole continuum of possible solutions for a particular task. In a machine learning setting, where one is acquiring a skill from demonstration, this raises the need for good generalization to solutions that possibly lie beyond the region of support of the original demonstration. Many existing data-driven approaches to humanoid motion synthesis are often limited in this respect - either they focus on interpolation within narrow regions near dense demonstrated samples or learning is posed as a problem of parameter tuning of an externally imposed path planning algorithm that may not naturally exploit the underlying structure of the space of solutions. We aim to make progress in this setting, by developing an algorithm that has better generalization properties and also a more natural and tighter integration between learning and planning.

In this setting, one way to obtain training data could be from demonstrated trajectories by an expert (Schaal et al., 2003). In this case notions such as optimality are

intrinsic to the expert's demonstrations and can be based on a variety of (sometimes unmodelled) factors (Coates et al., 2008). In order to have a better understanding of the behaviour of the algorithm, in this thesis, we utilize demonstration data that is obtained from another computational solution, involving numerical optimization with well defined cost functions. These solutions are computationally expensive and not feasible for online operation. However, they can serve the same role as demonstration data. With this, we have a clear idea of the specific optimality properties of each task being considered, and a measure of algorithm performance against reasonable 'ground truth'.

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., 2009b; 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.

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. Wang et al. (2008) introduced the GPDM, a Gaussian processes based dimensionality reduction with a dynamical model of the evolution of the state, that can learn models of human kinematic trajectories. In the same spirit, 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 parametrized path generation method over the learnt representation to generate novel movements.

Our goal is to learn a geometric structure, i.e., a skill manifold, that naturally and directly specifies both the low dimensional structure and evolution of trajectories on this subspace (which, in other works, one often externally and rather arbitrarily imposed). 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. We achieve this by representing motions in terms of skill manifolds (learnt from data) where the tangent spaces are suitably defined so that geodesics correspond exactly to the execution of the desired motion.

4.3 The manifold encoding

This section starts with a brief a summary of the manifold learning algorithm that was elaborated on in Chapter 3 and forms the basis of our method. It continues with the details of the procedure for generating trajectories on such learnt skill manifolds.

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. Popular examples include MDS (Hastie et al., 2001), LLE (Roweis and Saul, 2000) and ISOMAP (Tenenbaum et al., 2000). However, much of this work has been focused on summarisation, visualization or analysis that explains some aspect of the observed data.

On the other hand, we are interested in preserving properties of trajectories in the data set. So, our goal is to learn a model of the tangent space 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.

4.3.1 Learning the manifold model

This subsection presents a synopsis of the manifold learning method in the context of skill learning for robotic platforms. It concludes with a brief discussion on the parametrization of the skill.

A manifold encoding is grounded on a set of example solutions. Such a set is produced either from a computationally intensive optimization procedure, an optimal controller, or even a human demonstrator. The key characteristic is that such examples are solutions to a particular skill that are derived from the same underlying 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 in most cases represent trajectories of a particular skill. These are of the form,

$$\mathbf{Q} = [\mathbf{q}^1, \ldots, \mathbf{q}^k]^T, \tag{4.1}$$

$$\mathbf{q}^i = q_1^i, \ldots, q_n^i,\ i = 1, \ldots, k. \tag{4.2}$$

The number of examples being k and the length of each trajectory represented by i, the temporal ordering for each example trajectory. Each datapoint q_j^i 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 and has a specific geometry. 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, in contrast to task space representations that are much more well-behaved. In summary, the learnt skill-specific manifold is a hypersurface embedded in the ambient space, in this case the robot's configuration space, where all the set of the demonstrated skill solutions evolve.

We start by computing the neighbourhood relationships between datapoints of the demonstrated trajectories. The distance is computed with the Euclidean metric, while a threshold is automatically computed so that the datapoints belong to a single connected component. In addition a temporal relationship between consecutive points on each trajectory is enforced. A time-based sampling of the example solutions can have a negative effect on the model as it can skew the average distance metric that is estimated from the dataset. Consider for example a minimum jerk trajectory (Flash and Hogan, 1985) that is sampled in equal time steps. The datapoins that are near the start and end of the trajectory are points where the speed of the motion is low, thus producing a large number of samples. The cause of this effect is that a distance metric does not consider derivative information that datapoints carry. In this case a resampling is necessary that yields unit speed samples. In essence a bad estimate of this distance can either lead to overly dense geodesic paths, that impose an unnecessary computational burden, or to an overly sparse discretisation, that would not be able to follow the manifold geometry consistently. The average distance can be considered as a geodesic step estimate and is used to initialize the geodesic trajectory generation procedure, detailed in the next section.

Modelling $\mathcal{H}_\theta$ is done with a linear model of radial basis functions (RBF's) with features over the evidence (Hastie et al., 2001), where the number of basis functions, f, acts as parameter that can control the smoothness of the estimated mapping. More nonsmooth nonlinear manifolds with abrupt changes, would typically require more basis functions to ensure a tight local fit, though the generalization ability may be weakened. The RBF's, μ's and σ's are computed directly from the data, Specifically these are computed in an unsupervised manner using *k-means* clustering.

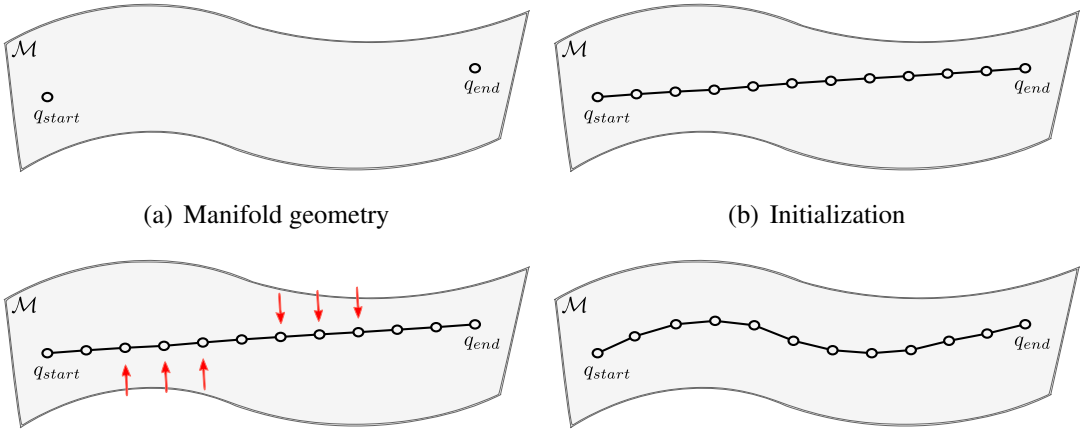
The free parameters of the model are only two and these are the dimensionality of the underlying manifold and the number of RBFs used in the encoding. Choosing the parameters affects the model complexity and falls under the general problem of bias-variance tradeoff. These two terms are not strongly coupled thus setting the parameters can be done through a cross-validation procedure. In this setting one would monitor the impact of the change of the parameters on the model error (Equation 3.1) and pick the least complex model that archives a good approximation. In essence we pick the model where the derivative of the error flattens.

4.3.2 Geodesic paths

By approximating the tangent space of the manifold, we gain access to a variety of geometric operations that allow us to generate novel solutions that conform to the manifold hypersurface. Central to our robotics aims is the ability to compute paths through configuration space that lie on the low dimensional manifold.

Formally, 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, discretise 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 4.1 presents sketches of each step of the trajectory generation procedure that is elaborated below.

The geodesic trajectory generation procedure consist 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. Note that the initial estimate is a discretised path (no need for forward integration) while the



(a) Manifold geometry (b) Initialization

(c) Gradient optimization (d) Geodesic path

Figure 4.1: 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.

Algorithm 2 Initial Geodesic Trajectory

INPUT: $\mathcal{M}$, ℓ, q_{start}, q_{end}

OUTPUT: $\mathbf{q} \equiv \{q_{start}, q_2, \ldots, q_i, q_{end}\}$

$q_{left} = q_{start}$, $q_{right} = q_{end}$, $dist_{Prev} = \infty$

loop

$diff = q_{left} - q_{right}$

$dist = norm(diff)$

if $dist < \ell$ **then**

break {Distance reached}

else if $dist < dist_{Prev}$ **then**

$midpoint = (q_{left} + q_{right})/2$

$q_{left} = InitialGeodesicTrajectory(\mathcal{M}, \ell, q_{left}, midpoint)$

$q_{right} = InitialGeodesicTrajectory(\mathcal{M}, \ell, q_{right}, midpoint)$

break

else

$dist_{Prev} = dist$

$q_{next} = q_{left}$ $(or\ q_{next} = q_{right})$

$H = \mathcal{H}_\theta(q_{next})$

$diff = H \times H' \times (diff/norm(diff))$

$q_{next} = q_{next} + diff \times \ell$ {Geodesic step}

end if

end loop

$\mathbf{q} = [q_{left}, q_{right}]$

model provides a continuous estimate of the tangent spaces of nearby points.

4.3.2.1 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 that 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 towards the middle.

Growing the path is done by 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 focus is equal or less to ℓ. Pseudocode of this phase is available in Algorithm 2 while Figure 4.1(b) provides a sketch of the outcome of the initialization step.

4.3.2.2 Geodesic path optimization

The initial estimate of the path is a manifold interpolation but roughly follows the manifold geometry and it not the shortest geodesic paths possible. Since we have learnt the tangent space of the manifold we can find an optimal – minimum cost – 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 of the gradient of

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

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

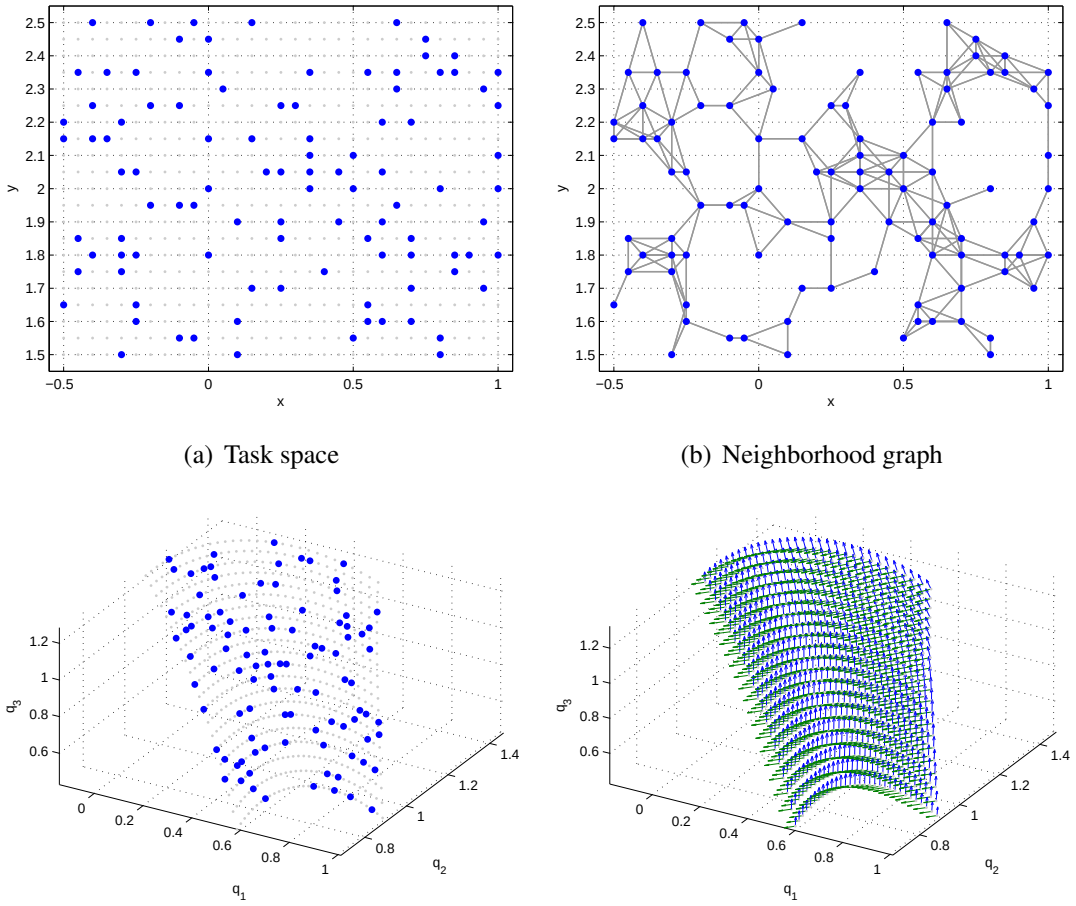
$$err_{length}(\mathbf{q}) = \sum_{i=2}^{n} \left\| q^i - q^{i-1} \right\|_2^2,$$

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

The next sections present three examples of our method. The first example presents experiments on a simulated 3-link arm where both the manifold and the learnt model can be visualized and are representative of the core ideas behind this work. For the second and third examples we use physical humanoid robots, with which we demonstrate how our method scales to more complex systems and more challenging tasks.

4.4 Reaching with a robotic arm

This experiment demonstrates how our framework can accurately learn and generalize a reaching behaviour. It shows how generated trajectories are consistent with the demonstrated examples while generalizing within and beyond the region of support of the data. We have chosen a *3-link planar* arm where we can explicitly visualize both the configuration space and the optimization manifold. The arm is a series of three



(a) Task space

(b) Neighborhood graph

(c) Joint space

(d) Tangent space

Figure 4.2: 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 neighbourhood graph used for learning a manifold. (c) The manifold hypersurface that we wish to encode, as traced by the dataset points in the configuration space of the robot. 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 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.

rigid links of unit 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).

4.4.1 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 4.2(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. The choice of the grid and the number of grid points are arbitrary and do not affect the subsequent process, later in this thesis we use a triangulated mesh to visualise the manifold geometry. 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.

The system being redundant, we first choose a redundancy resolution strategy, which implicitly specifies the manifold that we will subsequently learn. We choose the joint space configuration, $\mathbf{q}$, that minimizes the distance to a convenience (robot default or minimum strain) pose, $\mathbf{q}_c$:

$$\min \|\mathbf{q} - \mathbf{q_c}\|^2, \tag{4.3}$$

$$\textit{subject to } f(\mathbf{q}) - \mathbf{x} = 0, \tag{4.4}$$

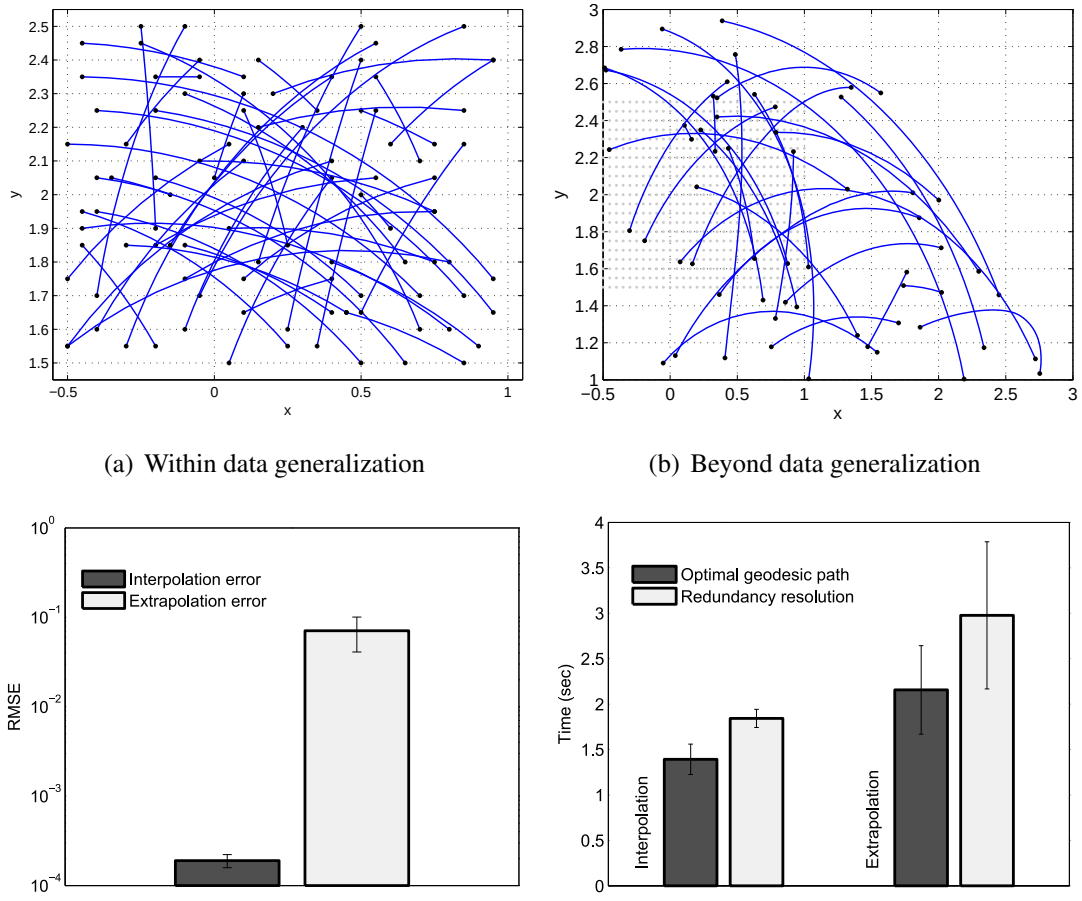
where f is the forward kinematics and $\mathbf{x}$ is the goal endpoint position on the plane.

The resulting $\mathbf{q}$'s trace a smooth nonlinear manifold in joint space, depicted in Figure 4.2(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 visualisable for more complex problems.

4.4.2 Implementation

The first step in data-driven learning of the desired manifold is to compute the neighbourhood graph of the training data. We evaluate the task space distances to compute the neighbourhood graph with the constraint that the graph contains a single connected component. In practice we gradually increase the neighbourhood distance until all points are connected, as in Figure 4.2(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 4.2(c). We



(a) Within data generalization

(b) Beyond data generalization

(c) Generalization errors

(d) Time

Figure 4.3: 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.

can subsequently evaluate $\mathcal{H}_\theta$ at any point in our joint space. Figure 4.2(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.

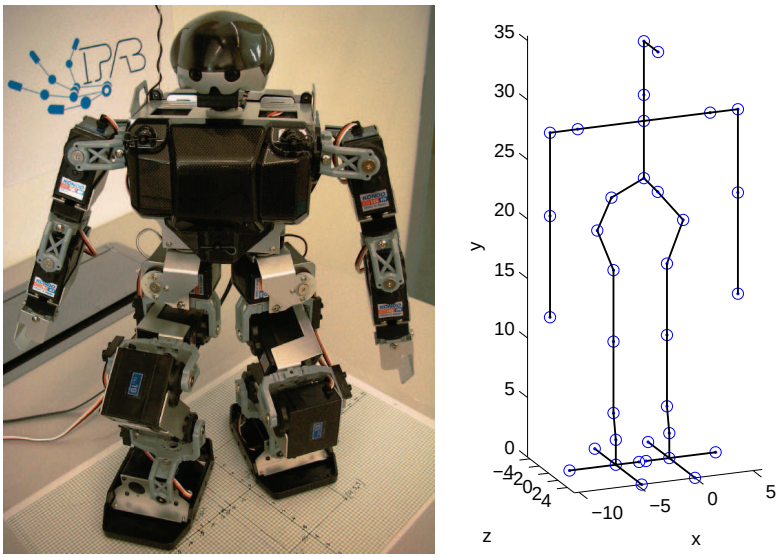
4.4.3 Generation of novel reaching solutions

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, when presented with novel planning queries. 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 geodesic paths, with random start and end points for each case, with the method detailed in section 4.3.2. Samples of such paths for both generalization cases are depicted in Figure 4.3(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 4.4.1, 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.

The averaged errors are depicted in Figure 4.3(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 $6.84 \times 10^{-2} \pm 2.19 \times 10^{-2}$. This shows that the manifold encoding is able to accurately represent and replace the original data generation procedure. In addition, computing the geodesic paths takes less time on average (Figure 4.3(d) in both cases).



(a) Robot (b) Model

Figure 4.4: The *KHR-1HV* humanoid robot used, (a) physical robot and (b) skeleton model.

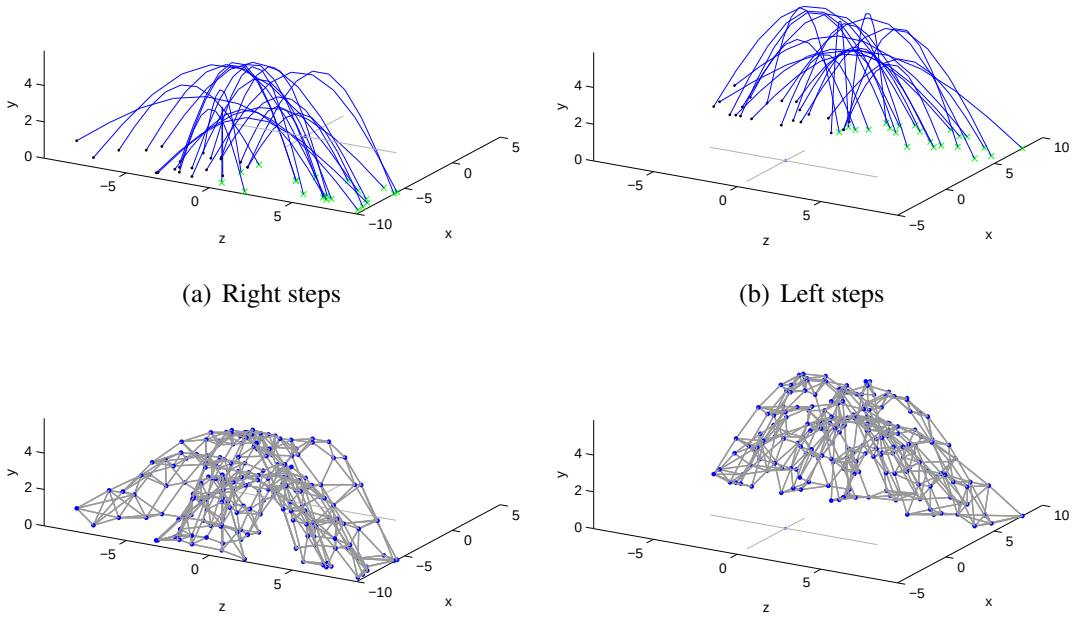
4.5 Walking with the KHR-1HV humanoid

The three-link arm experiments are useful for demonstrating the working of the manifold learning and geodesic path planning algorithm. We now move to a more complex system. In this setting, the notion of a skill manifold is more intuitively understood. We use the *KHR-1HV* (Figure 4.4(a)), a "KidSized" humanoid robot[1] that stands approximately 35cm tall. It consists of 19 digital servo motors on brackets, in a bipedal-two-armed configuration, with a control board and a battery pack. The system is unstable as the center of mass is elevated.

No analytical model of the dynamics of the system is available to us. Obtaining such models is labour intensive and requires detailed knowledge of the structure of both the robot and the environment. Building such models require system identification and parameter estimation methods that are subjects of current research work. Robotic systems are hard to approximate or analytically model, thus we prefer to work directly from experimental data.

We focus on the task of walking, with the aim of generating a motion synthesis strategy that allows for full coverage of a reasonably large interval in step length. Recalling the Nao goalkeeper example, such a skill manifold would allow the robot to

[1] According to the RoboCup Humanoid League size classification.



(a) Right steps

(b) Left steps

(c) Right neighborhood graph

(d) Left neighborhood graph

Figure 4.5: *Task space* representation of the training data through forward kinematics. Random start and end point leg swing trajectories of the left (a) and right (b) legs. (c) and (d) the neighbourhood graphs that result from the task space distances between demonstrated data (units in *cm*). This provides the task-specific distance metric for the high dimensional *joint-space*. Note that depicted here are only feet midpoint positions while the datasets consist of the joint space points that are 19-dimensional.

take large steps when trying to reach the ball, and smaller steps when near the ball or when correcting its orientation with respect to a team-mate for passing, in a continuous fashion. We begin with a redundancy resolution strategy that would yield training data and ground truth for our subsequent comparisons.

4.5.1 Example walking solutions

We frame the redundancy resolution strategy as an unconstrained nonlinear optimization problem. Algorithmically, we use a Quasi-Newton approach with a cubic line search procedure, based on the BFGS formula for iteratively updating the estimate of the Hessian of the objective (cost) function Nocedal and Wright (2006). Formally, the optimization problem is of the form

$$\min_{q} \mathcal{J}(\mathbf{q}), \tag{4.5}$$

$$\textit{subject to } f(\mathbf{q}) - \mathbf{x} = 0, \tag{4.6}$$

where $\mathcal{J}$ is the cost function, f is the forward kinematics and $\mathbf{x}$ is a goal task space position. The cost function is a mixture of task constraints and stability constraints. The cost function evaluates:

- the distance of the midpoint of the swing foot to the desired goal
- the alignment of the swing foot with the x and y versors, to keep the foot flat
- the horizontal distance of the position of the pelvis to the desired pelvic position, to manipulate the center of mass of the humanoid
- the alignment of the waist of the robot with the z versor, to keep the humanoid, from the hips up, in an upright position

The optimization initialization pose is one where the humanoid stands upright with the knee joints slightly bent.

To generate a walking trajectory we start with the desired task space path of the swing leg and the position of the pelvis, and discretise to 20 waypoints. The swing foot trajectories are straight lines from start to goal points while the height of the foot is regulated with a sinusoid, scaled to a prespecified 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. Last we run the optimization procedure detailed 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 20 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. Figure 4.5(a) and 4.5(b) show the task space trajectories of each swing leg by running the datasets through the forward kinematics (the support foot is in light gray for comparison).

4.5.2 Implementation

Compared to our previous simpler example, this is a higher dimensional space and sampling is necessarily somewhat sparse. Of the 19 DoFs of the robot we used the 12

DoFs of legs and hips and kept the remaining arm joints at a constant pose. Furthermore we separated each footstep to a swing phase and a weight shift phase. This way we divided learning into two components, leg swing manifold and support weight shift manifold, as the measure of optimality is essentially different for each phase. In general the separation of such motions can be automatically done, for example Beaudoin et al. (2008), but this topic is beyond the scope of this thesis.

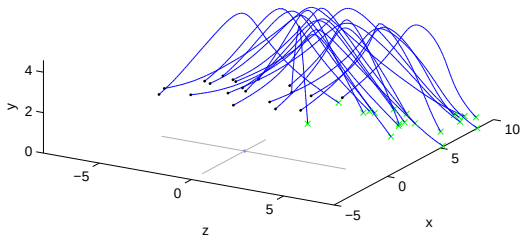
We begin with the same neighbourhood graph computation procedure where we gradually increase our neighbourhood distance until the graph is not disconnected (Fig 4.5(d) and 4.5(c)). We set the dimensionality of the manifolds to be 3, corresponding to the natural task space of the robot (see section 4.7). In all learnt manifolds we used models with 20 RBF's and 400 data points that belong to 20 random task space trajectories as described in the previous section.

4.5.3 Generation of novel walking motions

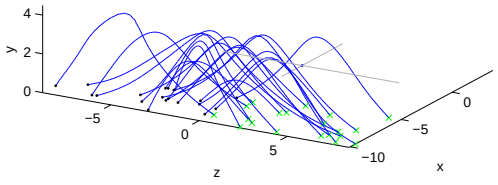
The learnt manifolds are able to produce smooth walking trajectories that satisfy the optimization criteria used to produce the training data. Specifically, the average *RMSE* (*degrees*) of the leg swing manifold for the ground truth was as low as 0.12 while the average *RMSE* of the weight shift manifold ranged on average near 0.06 (Figure 4.6(c)). This implies that the geometry of the step manifold is more complex and some of its features might be smoothed over by the RBF model. Nonetheless the procedure was able to produce stable walking in the continuum of the reaching space of the robot as depicted in Figure 4.6(a) and 4.6(b) for right and left swings accordingly.

One point to note is that the shape of the trajectories in task space is qualitatively different than the training data. This suggests that the learnt manifold indeed traces the true underlying geometry that the optimization procedure sculpts in the robot's joint space. In contrast the training data has been generated on a point by point basis, while the shape of the trajectories in the task space (sinusoid) has been artificially imposed, regardless of the intrinsic structure of the optimality surface. The geodesic paths that are generated are optimal with respect to the manifold's geometry and traverse the configuration space smoothly.

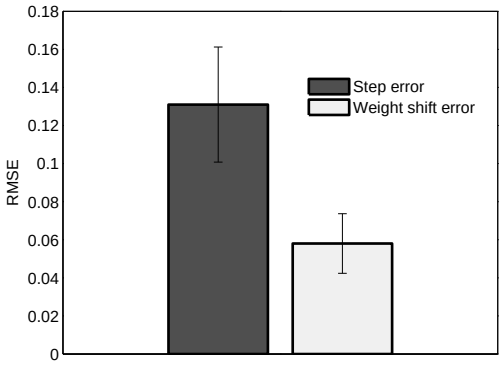
The absolute time needed to generate a geodesic path on the pair of manifolds (swing leg and weight shift) from random start to random end points was approximately 1.5552 ± 0.4785 seconds (in a standard, not particularly fine-tuned, numerical implementation of the algorithm) whereas generating a trajectory with the optimization



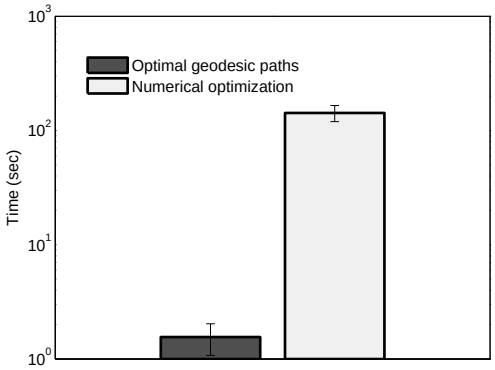
(a) Generated left steps



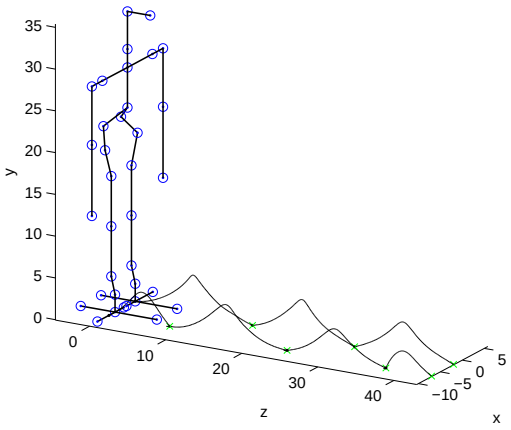
(b) Generated right steps



(c) Error



(d) Absolute time



(e) Generated random walk

Figure 4.6: Experimental results with the humanoid robot. Random start and end point trajectories for left (b) and right (b) leg swings that have been generated from our learnt manifold, via geodesic path optimization (units in *cm*). (c) *RMSE* (*degrees*) of generated data against ground truth. (d) absolute time needed for planning and optimization with our method and the nonlinear optimization method (y axis in logscale) described in the text (section 4.5.1). (e) Random walk generated by geodesic path optimization on the learnt manifolds for randomized task-space goals. Snapshots of the robot executing the motion in Figure 4.7, see also accompanying video.

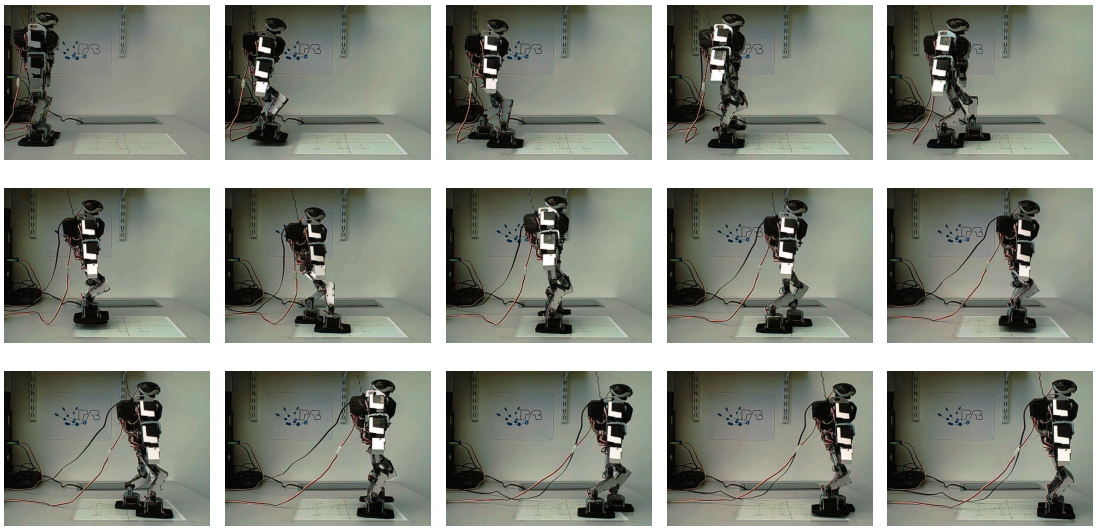


Figure 4.7: Stills of the robot executing the planned motion depicted in Figure 4.6(e).

procedure, described in section 4.5.1 required approximately *two minutes* on average. This is a *significant* decrease in absolute planning time, which makes it possible to deploy this algorithm in realistic application scenarios (e.g. RoboCup).

A randomized walk sequence entirely generated with our method is depicted in Figure 4.6(e). Notice that the step lengths are varying and the step points are variable as well with respect to the x axis. Snapshots of this walk executed by the robot are shown in Figure 4.7.

4.5.4 Experimental considerations

In this section, we make a few practical observations regarding the humanoid platform used in the experiments. The intent is to give an idea of some limitations that are specific to this robot - independent of the capabilities of the algorithm - while we later move to more capable experimental platform. Throughout the course of our experiments we had to deal with various shortcomings of the platform. One of the shortcomings of the KHR platform is that the servos are not strong enough to support the robot's weight when in single foot support. So, this creates oscillations while walking and makes leg placement less accurate. Also, it is worth noting that the robot tends to slide the swing leg before reaching the goal as the support servos give in.

In addition the control of the servos is very limited. On that we can solely command servo positions and maximum velocity while torque control is implemented in an uncontrolled lower level. Essentially this leaves room only for open loop control,

both on the trajectory and the joint level as no feedback is available from the robot servos, an approach that is not suited for unstable dynamical systems. Closed loop control would be far more efficient for dynamical tasks especially for bipedal motions that require active balancing strategies. We believe that also solutions of a dynamic query with regard to a specific skill, e.g. dynamic walking, should also trace a non-linear manifold. The investigation of this claim is in fact part of our current research focus.

Furthermore, for learning in a humanoid setting ideally one would utilize human motion data. The difficulty with this would be to map such data to the robot geometry which is an active research thread, currently beyond our scope.

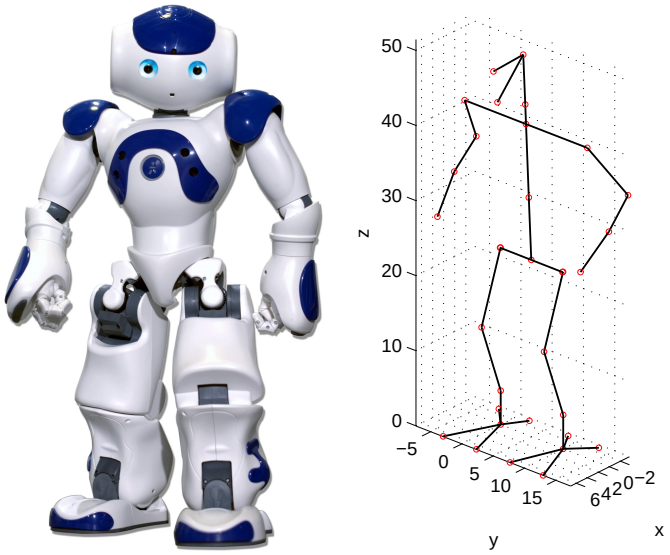
4.6 Walking with NAO humanoid robot

This section presents how the framework can be used with an advanced 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 walking motions. Furthermore, the produced motions can accommodate novel starting positions and goals while producing good (stable) generalizations of the examples provided.

This case allows us to demonstrate an intuitive example of a skill manifold. Our experiments are based on the *Nao* (Figure 4.8) 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.

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. 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.

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



(a) Robot (b) Model

Figure 4.8: The Nao humanoid robot used, (a) physical robot and (b) skeleton model.

that guides the path in that particular neighbourhood. We begin with a redundancy resolution strategy that would yield walking examples as training data for manifold learning.

4.6.1 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 (Nocedal and Wright, 2006).

The optimization problem is of the form,

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

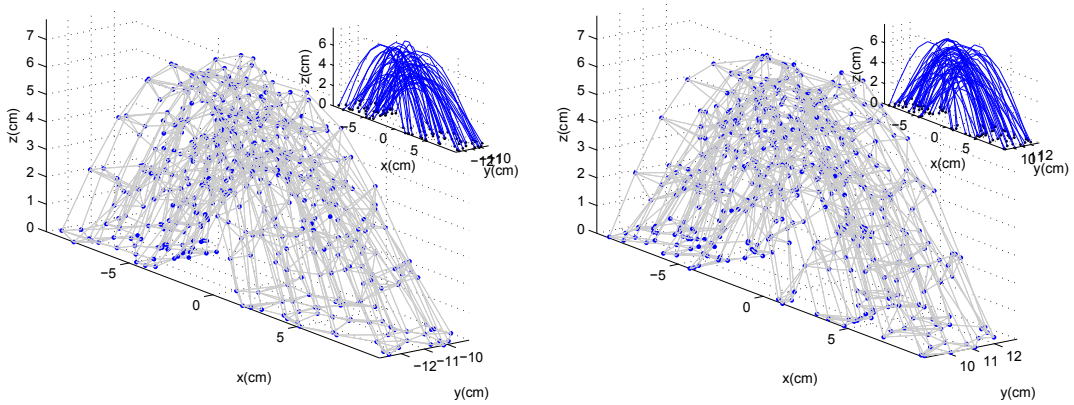
$$\textit{subject to } f(\mathbf{q}) - \mathbf{x} = 0,$$

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:

- distance between swing foot and goal
- alignment between swing foot and x/y versors

- deviation in pelvis position
- alignment between waist and z versor

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



(a) Right foot training set

(b) Left foot training set

Figure 4.9: The neighbourhood 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.

To generate a walking trajectory we start with the desired task space path of the swing leg and the position of the pelvis, and discretise 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. For stability control 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 4.9(b) and 4.9(a) show the task space trajectories of each swing leg foot midpoint, by running the datasets through the forward kinematics of the system.

4.6.2 Implementation

Compared to our example in section 4.4, 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.

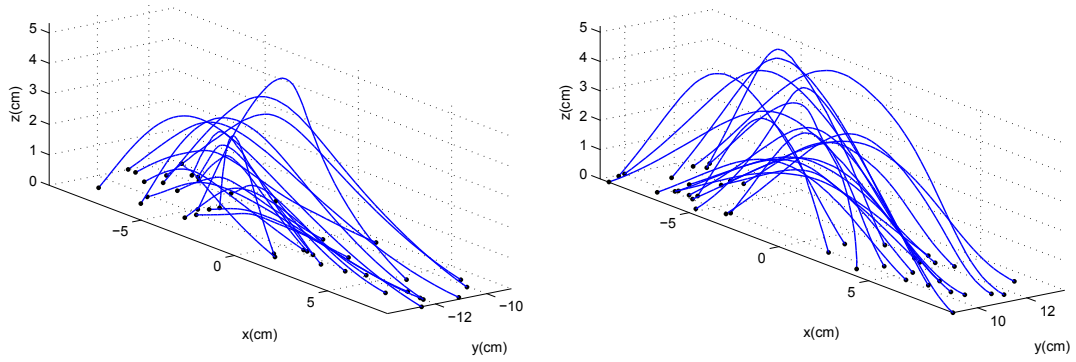
We begin with the same neighbourhood graph computation procedure where we gradually increase the neighbourhood distance until the graph is not disconnected (Figure 4.9(b) and 4.9(a)). We set the dimensionality of the manifolds to be 4, with a simple cross validation step that penalizes model complexity 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.

4.6.3 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 4.11), 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 4.10(a) and 4.10(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 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



(a) Generated right foot traj. (b) Generated left foot traj.

Figure 4.10: Generated 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.

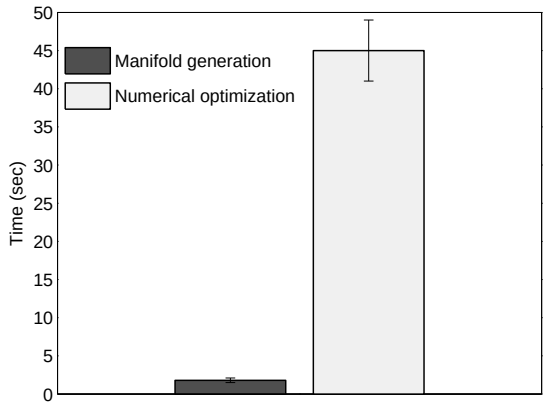


Figure 4.11: Results averaged over 25 random start-to-end trajectories. The manifold representation can stepping trajectories that generalize from the numerical optimization examples in significantly less time. This way the manifold can been employed in an *online* scenario while producing trajectories that, with a numerical approach, would take close to a minute to compute.

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.

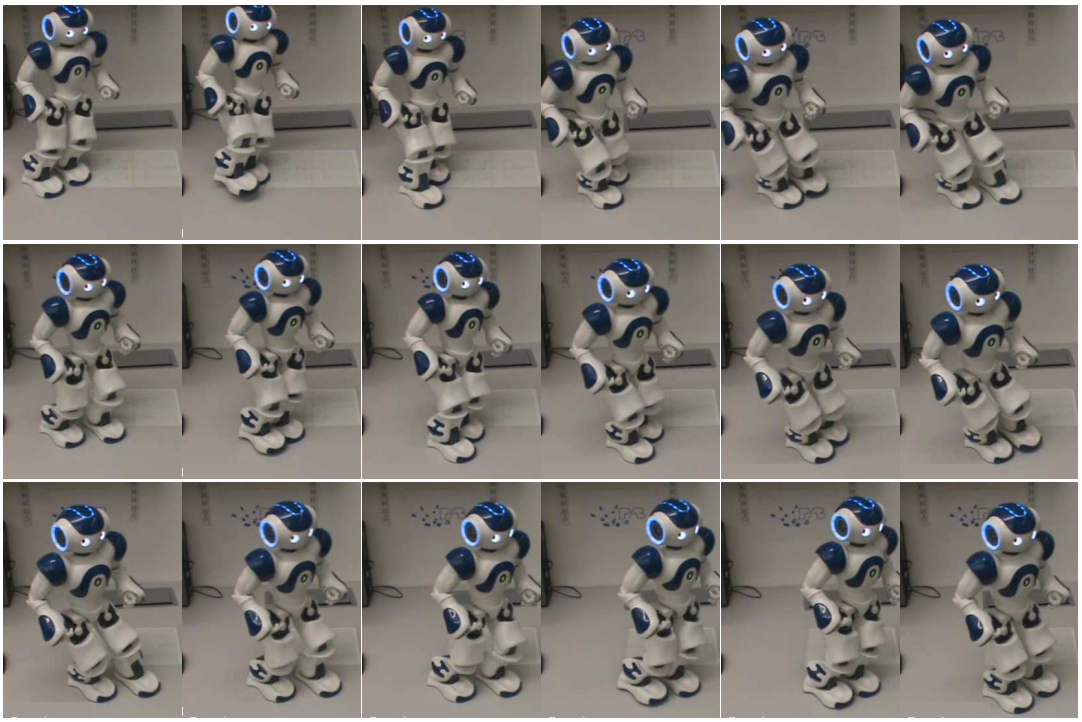


Figure 4.12: An example of a generated walk from random feet start and end-point positions. The generated walk is stable and relatively fast while the possible reach of the steps is greater than the standard motion synthesis procedure from Aldebaran Robotics. Additionally the motion generation is fast and can be employed in an online motion planner.

4.7 Discussion

We have demonstrated how a machine learning technique for approximating a low-dimensional skill manifold may be tightly integrated with the process of trajectory generation. One of the important differences between our manifold learning framework and other machine learning-based motion generation algorithms is that we focus on capturing the structure underlying the demonstrated set in a geometric fashion.

In both examples presented, we have chosen d to have the dimensionality of the system's task space. The reasoning behind this choice is that there might be configurations that are close in joint space but far away in task space. Since our aim is to

learn skill-specific manifolds, this seems natural.We could have used any $d < D$, but simpler models are preferred. Choosing the appropriate dimensionality falls under the bias-variance trade off, as discussed below.

We now make a few observations regarding limitations (hence, directions for future improvement) of the algorithm in its current form. In this work, we do assume that the skills may be represented by a subspace that is a single connected component. This is clearly not an issue for the 3-link arm example. However, in general, this may well be insufficient as the dimensionality of the system grows. The place where this plays a role is the neighbourhood graph computation where by connecting two points that should not be connected we would obtain a skewed model. In practice, suitably dense sampling, or better still incremental sampling in appropriate regions, and a bit of algorithmic book keeping, would suffice to ensure that this aspect of the manifold structure is properly reflected.

Also, one must keep in mind that the manifold learning step is performed with an iterative algorithm, much like EM, that is randomly initialized and does not always guarantee a global minimum. So, learnt models may not be unique solutions. This may call for better model selection procedures - a topic for future development.

The number of RBF basis in our experiments was chosen empirically, thus is open to further improvement. A high number of RBF's would allow the model to capture more intricate local geometric structure of the manifold, but would impair its generalization ability. On the other hand a low number of RBF's may oversmooth the solution and lose much of the geometric variation present in the training data.

The above issues derive from the bias-variance trade-off, central to machine learning, and can be handled systematically with the use of cross-validation procedure. Such choices would need to be closely related to the geometric complexity of the manifold that one would like to learn. Also, by using the centred estimate of the directional derivatives as a basis for our model formulation, it is implied that the model can have limited success with manifolds that cannot be locally approximated in a quadratic form. In practice highly nonlinear manifolds that vary wildly or have sudden cutoffs may not be suitable for learning, without additional treatment.

Finally, we assume that start and end points of each trajectory are known. For this we have used the redundancy resolution strategy used in generating the demonstrated data. There is no implicit mapping of task space goals to configuration space poses on the manifold per se, but in principle once the manifold is learned one can easily search for points that satisfy task space goals.

4.8 Conclusions

We have demonstrated how a manifold learning algorithm can capture the geometric properties of a low dimensional skill manifold that underlies a high dimensional dataset. We have also shown how this model can be naturally used to generate joint space trajectories, and how the generated trajectories reflect the optimality and constraints inherent in the training data.

We started with an example of a simulated robotic arm that is suitable for demonstrating the core concepts of our work and then demonstrated a similar result on a more interesting humanoid robot behaviour. We have demonstrated how manifolds of complex numerical optimization solutions can be learnt from sparse data and how the geometric structure generalizes within and beyond the support of the data. Finally, we have shown how such learnt manifolds can be used to produce novel approximately optimal solutions to continuous path planning queries in a very efficient and fast manner.

The next chapter chapter builds on the geodesic trajectory generation method that we have presented and extends it for planning in the presence of kinodynamic constraints. It shows how such geodesic trajectories can be modified “on-line” to accommodate a changing environment while reusing an existing skill manifold encoding.

Chapter 5

Geodesic trajectories with dynamic constraints

This chapter addresses the problem of compactly encoding a continuous family of trajectories corresponding to a robotic skill, and using this representation for the purpose of *constrained* trajectory generation in an environment with many (possibly dynamic) obstacles. With a skill manifold that is learnt from data, we show that constraints can be naturally handled within an iterative process of minimizing the total geodesic path length and curvature over the manifold. We demonstrate the utility of this process with two examples. Firstly, a three-link arm whose joint space and corresponding skill manifold can be explicitly visualized. Then, we demonstrate how this procedure can be used to generate constrained walking motions with the Nao humanoid robot.

5.1 Changing environments and dynamic constraints

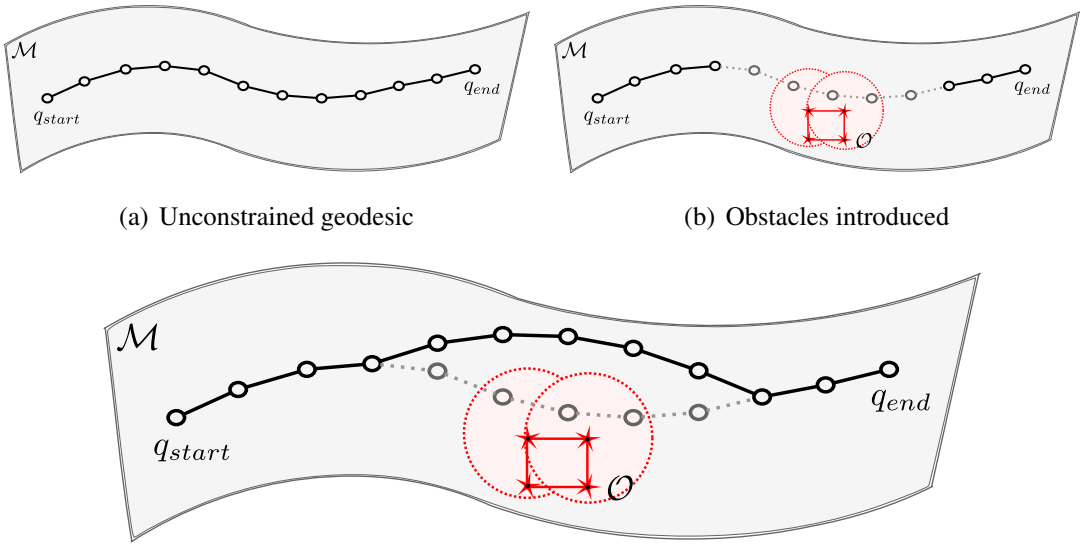
Humanoid robots receive increasing attention as general purpose platforms suitable to a multitude of applications. However, the level of flexibility that can actually be achieved tends to fall short of this promise of generic dexterity. One of the big difficulties is related to the problem of devising motion planning and control algorithms that can cope with the combination of dynamic complexity, dimensionality and model imprecision. Many off the shelf solutions providing humanoid behaviours, e.g., for locomotion, tend to be restricted to a limited and discrete vocabulary, valid only in narrow domains of applicability. For instance, it is hard to find a general purpose humanoid walking 'engine' that provides full control over step length, width and height, in real-world terrains. On the other end, specialized approaches that do enable some flexi-

ble movements tend to be computationally expensive, e.g., requiring high-dimensional numerical optimization and/or c-space search, often with near-exact knowledge of the system and its environment. This is a steep requirement for resource constrained machines.

In this chapter, we address the problem of designing motion strategies that can achieve a rich set of within-skill variations. These strategies are acquired in a data-driven manner, allowing for adaptation to changes in environmental conditions and task contexts, and can be implemented in realistic resource constrained robotic systems. Such a representation of a parametrized skill, applicable under a wide variety of conditions, could then form the basis for higher level search processes over a small alphabet, enabling fast high level planning (e.g., Chestnutt et al. (2005); Beaudoin et al. (2008)). Indeed, one of the big weaknesses of some existing motion synthesis strategies is that they do not cleanly admit such abstractions (Calinon and Billard, 2008; Gribovskaya et al., 2010; Wang et al., 2008; Chalodhorn et al., 2006; Bitzer et al., 2008).

We build on the work presented in the previous chapter – to compactly represent a continuous family of trajectories representing a specific skill such as variable step-length walking – to incorporate constraints (involving a combination of task and joint space obstacles). The goal is to define a scheme wherein the manifold captures the essential variations in the set of trajectories corresponding to a skill, from which one is able to lazily select specific instances as the constraints (possibly dynamic) are revealed. In practice, one often adopts a receding-horizon approach to handling such problems and we'd like our approach to be compatible with this paradigm.

The basic notion of utilizing low-dimensional representations in a motion synthesis setting is becoming well accepted in the robotics and graphics communities (Ramamoorthy and Kuipers, 2008; Isto and Saha, 2006; Safonova et al., 2004). Some recent works (Berenson et al., 2009b; Stilman, 2007; Bretl et al., 2004) 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, this requires full access to an exact model, which may not always be possible. The machine learning literature includes many examples of dimensionality reduction methods used to abstract and/or make problem spaces manageable. Wang et al. (2008) introduced the GPDM, which identifies a mapping to a low dimensional space where a linear dynamic model is fit to data. In the same spirit, Bitzer et al. (2008) use a Gaussian Process-based nonlinear dimensionality reduction technique to arrive at a



(a) Unconstrained geodesic

(b) Obstacles introduced

(c) Constrained geodesic path

Figure 5.1: 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 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, O, taken into consideration serves to drive the geodesic trajectory away from the red square obstacle, subject to an obstacle clearance distance ℓ. This is achieved through a *spring-system* model that extends the geodesic trajectory generation procedure and produces solutions that avoid such "*no-go*" patches while following the underlying manifold geometry.

subspace within which one may approximate demonstrated data using parametrized families of paths. An approach that is closer to our current work is that of Calinon and Billard (2009), who demonstrate robot programming by demonstration with a probabilistic model, namely Gaussian Mixture Regression, of Jacobian-based inverse kinematics for learning trajectories and incorporating task space constraints. What has not always been exploited in such work is the *geometrical structure* of families of paths in lower dimensional subspaces. In essence by looking at the low dimensional space as a geometrical structure that is embedded in the ambient space of the system we gain access to a number of efficient planning, replanning and control methods. In contrast, by simply mapping a set of poses to a low-dimensional space and fitting a parametrized model, one is essentially *overriding* potential intrinsic dynamics effects that define

many behaviours of interest.

Our goal is to learn this 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*.

Figure 5.1 provides a sketch of such a scenario. In this setting, we can generate unconstrained trajectories of the learnt skill manifold, as explained in the previous chapter (Figure 5.1(a)). Now as the system's environment changes, a set of novel obstacle points is introduced. What our method achieves, is to generate geodesic trajectories that can successfully navigate away from such obstacle sets while not leaving the support of the learnt manifold (Figure 5.1(c)). This is achieved with a spring-system model that optimizes geodesic trajectories with respect to novel constraints.

The next section explains the constrained geodesic trajectory generation procedure and elaborates on the spring-system model that has been employed. The following section demonstrates how such an approach can be applied to a 3-DoF arm that performs a reaching task, and to the Nao humanoid robot performing walking motions.

5.2 Novel constraints on learnt manifolds

Presented with D-dimensional data that are derived from a d-dimensional geometry embedded in the high dimensional space, we can learn a representation of such a geometry with the tools we have developed in the previous chapters (Havoutis and Ramamoorthy, 2010b). The manifold learning method we have presented in Chapter 3 provides us with a global model of the underlying hypersurface's tangent basis, $\mathcal{H}_\theta$, which serves as a compact encoding of the manifold geometry.

Given such a model we are then able to produce unconstrained solutions to novel motion planning queries as geodesic trajectories on the learnt manifold. This generative procedure has been explained in Chapter 4. The outcome is a series of points in the system's state space, $\mathbf{q} = [q_{start}, q_2, \ldots, q_{end}]$, that follow the manifold's geometry that in turn encodes task and system costs and constraints that where present in the set of solutions utilized as a training dataset in the model learning phase.

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 geometries 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.

Re-learning a representation that takes account such randomly occurring constraints would be futile. Instead, our method reuses the previously learnt unconstrained manifold representations and incorporates new constraints in the motion planning phase, in an on-line manner. The next subsection describes the procedure in detail.

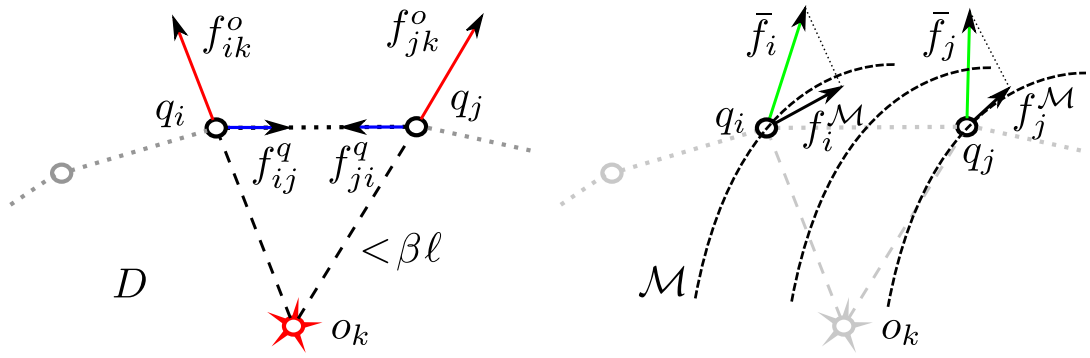
5.2.1 Constrained geodesic paths

The algorithm presented in subsection 4.3.2 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 $O \in \mathbb{R}^D$ 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 obstacle or a set of points sampled from the surface of a sphere (Figure 5.1(b)).

The intersection of the manifold set and the obstacle set, $\mathcal{M} \cap O$, 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 we want to avoid but given the learnt tangent space the path will not leave the surface of the manifold, thus the optimality 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 neighbours. A force f_{ij}^q, with magnitude

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



(a) Calculation of forces (b) Manifold projection

Figure 5.2: (*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 $\mathcal{M}$.

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 (subsection 4.3.2). The magnitude of the force is directly proportionate to the distance of the points in question.

The obstacle point set, 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 $\beta\ell$. This distance can also be increased or decreased by tunning the scalar β with obstacle clearance in mind.

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 projected on 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. Figure 5.2(b) provides a sketch of the average force projection step. In effect this makes the points take small geodesic steps away from the

Algorithm 3 Constrained Geodesic Trajectory Generation

INPUT: $\mathcal{M}$, q_{start}, q_{end}, O

OUTPUT: $\mathbf{q} \equiv \{q_{start}, \ldots, q_i, q_{end}\}$

$\mathbf{q} \leftarrow$ Geodesic Path(q_{start}, q_{end})

repeat

$d^O \leftarrow$ Compute Distances($\mathbf{q}$, O)

$[f_{ik}^o, f_{ij}^q] \leftarrow$ Calculate Spring Forces($\mathbf{q}$, d^O)

$\bar{f}_i \leftarrow f_{i.}^o + f_{i.}^q$

$\bar{f}_i^{\mathcal{M}} \leftarrow \bar{f}_i \mathcal{H}_\theta^{\mathbf{q}} \mathcal{H}_\theta^{\mathbf{q}'}$

$\mathbf{q} \leftarrow \mathbf{q} + \gamma \bar{f}_i^{\mathcal{M}}$

$\mathcal{C}^i \leftarrow \partial \mathbf{q}^2 / \partial s^2$ {Curvature}

$\bar{\mathcal{C}} \leftarrow 1/n \sum \mathcal{C}^i$

$\bar{\mathcal{C}}_i^{\mathcal{M}} \leftarrow (\bar{\mathcal{C}} - \mathcal{C}^i) \mathcal{H}_\theta^{\mathbf{q}} \mathcal{H}_\theta^{\mathbf{q}'}$

$\mathbf{q} \leftarrow \mathbf{q} + \gamma \bar{\mathcal{C}}_i^{\mathcal{M}}$

$\delta \leftarrow \mathbf{q} - \mathbf{q}_{old}$

until $d^O = \{\oslash\}$ *or* $\delta \leq 10^{-6}$

obstacle while not leaving the support of the manifold. We repeat the procedure until all path points have cleared obstacle points ($O = \{\oslash\}$) or the algorithm has converged.

5.2.1.1 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.

Definition 5 ({Geometric} Curvature)**.** *A measure of the speed of rotation of the tangential frame. Intuitively, (geometric) curvature is the amount by which a geometric object deviates from being flat, or straight in the case of a line.*

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, $\mathbf{q} = q_1, \ldots, q_n$ as

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

where s is the distance between two consecutive points. We calculate the mean curvature $\bar{C}$ and the error gradient $\bar{C} - C^i$ (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. The entire constrained geodesic trajectory generation procedure is summarized in Algorithm 3.

The following sections present two examples of our method. The first example presents experiments on a simulated 3-link arm where both the manifold and the learnt model can be visualized and are representative of the core ideas behind this work. For the second example we use a physical humanoid robot, with which we demonstrate how our method scales to more complex systems and more challenging tasks.

5.3 Constrained reaching on 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).

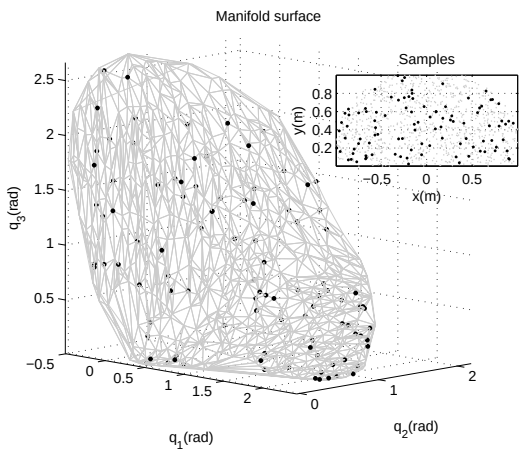
5.3.1 Reaching examples

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 couples, where

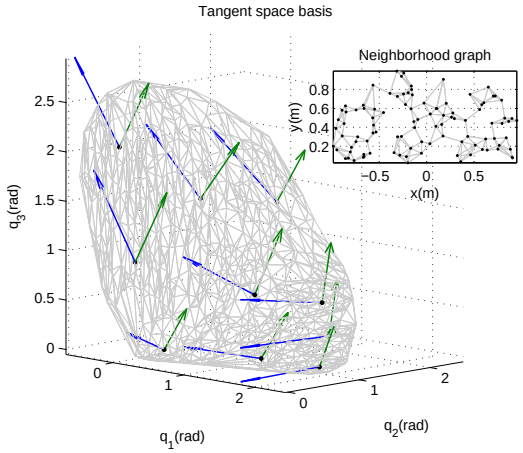
$$\mathbf{x} = \begin{cases} x \in [-1, 1] \\ y \in [0, 1] \end{cases} \tag{5.1}$$

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 5.3(a)), as black dots in joint space and task space plots.

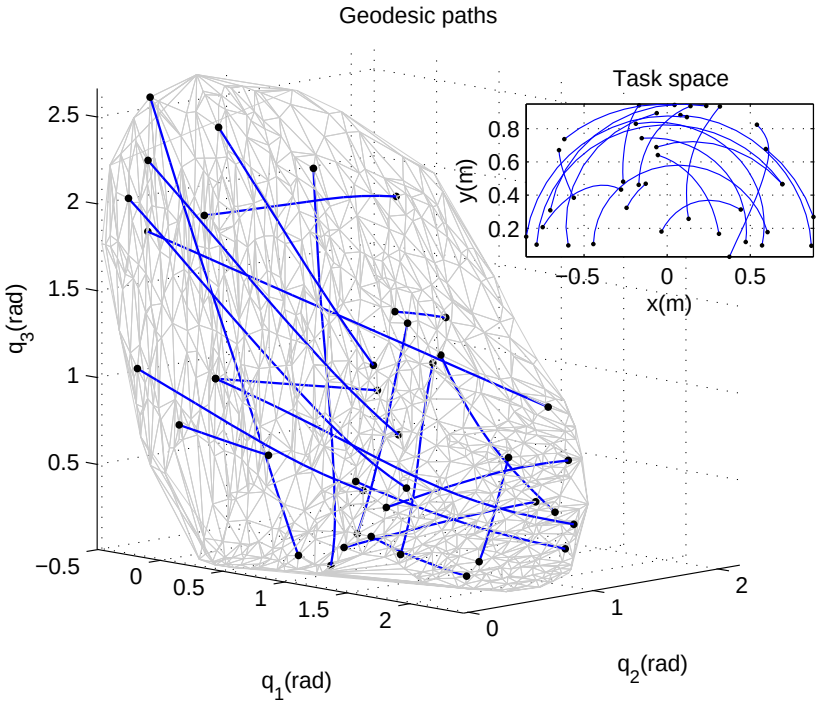
We densely sample the space with 900 more points that are used solely for visualization purposes and play no further part in the learning procedure. For visualization purposes, we use all 1000 points to compute a Delaunay triangulation of the joint space structure as sampled, and then plot the trimesh (triangle mesh) for comparison with the



(a) Samples.



(b) Learnt manifold.



(c) Generated geodesic paths.

Figure 5.3: 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 neighbourhood graph in task space (inset plot), and the learnt tangent space that the model predicts for the RBF centres 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.

paths that our algorithm produces. This trimesh surface is depicted in all figures with thin gray edges.

The system being redundant, we first have to choose a redundancy resolution strategy, which implicitly specifies the geometry of the manifold (Figure 5.3(a), thin gray mesh) that the manifold encoding approximates. 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,

$$\min \|w\mathbf{q} - \mathbf{q_c}\|^2, \tag{5.2}$$

$$\textit{subject to } f(\mathbf{q}) - \mathbf{x} = 0, \tag{5.3}$$

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 (Craig, 1989; Siciliano and Khatib, 2008; Levine, 1996) 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 systems weight as well.

The resulting $\mathbf{q}$'s trace a smooth nonlinear manifold in joint space, depicted in Figure 5.3(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 visualisable, for many complex problems.

5.3.2 Implementation

We start by computing the neighbourhood graph of the dataset points. We do this by evaluating the task space distances as the forward kinematics of the system are known. As we require that our set consists of one connected component, we gradually increase the neighbourhood distance until no disconnected subsets exist. The resulting neighbourhood graph is depicted in Figure 5.3(b)(inset plot).

After visualizing the optimality surface we can conclude that the manifold can be naturally represented with a two dimensional tangent space, and we learn a model of

$\mathcal{H}_\theta$ with 10 RBFs. We can subsequently evaluate $\mathcal{H}_\theta$ at any point in our joint space. For example Figure 5.3(b) shows the tangent basis evaluated at the centres 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 5.3(b) for each point in question.

5.3.3 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 geodesic paths. The procedure for generating these unconstrained paths was described in section 4.3.2, and the key advantage is that the generated paths will be of shortest distance and adhere to the manifold geometry. Geodesic paths generated from randomly sampled start and end points are depicted in Figure 5.3(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 interesting observation discussed in the next subsection.

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 5.2.1, and results are depicted in Figure 5.4. We start with a set of random start and end points and pick a list of obstacle points that intersect the manifold. In Figure 5.4 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.

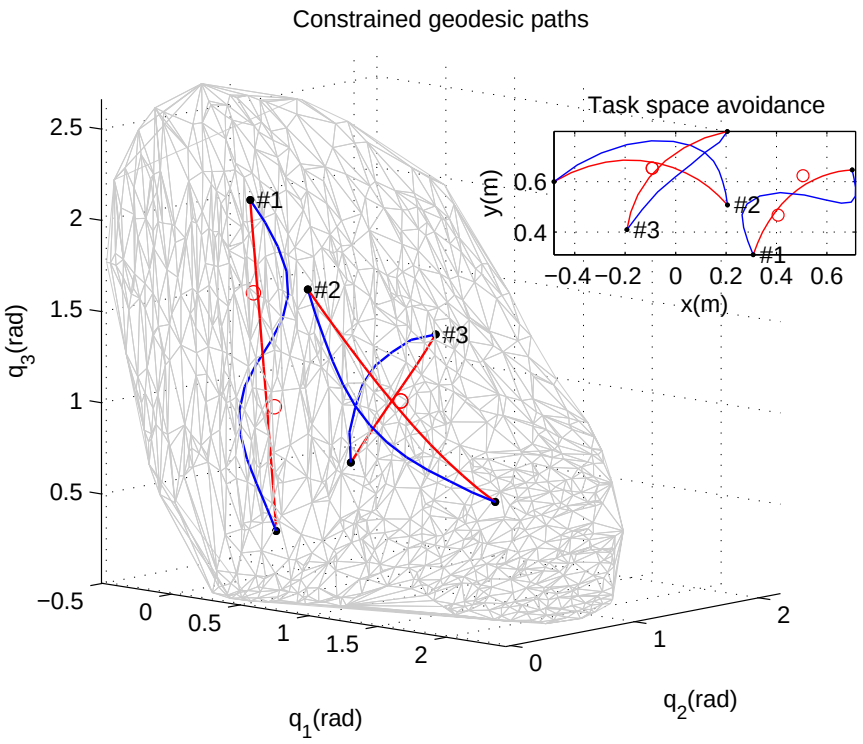


Figure 5.4: Example geodesic trajectories that avoid point set obstacles on the learnt manifold. The unconstrained geodesic trajectories in red are the initial estimates that either collide or come unsuitably close to the obstacle points. The blue trajectories are the outcome of optimizing these geodesic 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 appear in the system's task space. See text for details.

5.3.4 Remarks

One interesting 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 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 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 5.3.1, 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.

5.4 Constrained stepping with the Nao humanoid

The three-link arm experiments are useful for demonstrating the working of the manifold learning and constrained geodesic path planning algorithm. Now we show how our proposed approach can be applied to a more complex setting. This allows us to demonstrate an intuitive example of a skill manifold. Our experiments are based on the *Nao* (Figure 4.8) humanoid robot, previously presented in section 4.6.

The following set of experiments aim to demonstrate that first our method can accurately represent a complex bipedal humanoid behaviour, such as walking, that requires a high degree of coordination and is subject to a number of stability constraints, and second that the method of adding novel constraints to the trajectory generation procedure scales up to a system of such dimensionality. We demonstrate this with a walking task. Our aim is to generate a motion synthesis strategy that achieves full coverage of a reasonably large interval in step length, width and height, while allowing the addition of constraints that appear as the system's environment changes. We begin with a redundancy resolution strategy that would yield walking examples as training data for the manifold learning procedure.

5.4.1 Stepping examples

As in 4.6.1 we generate a set of randomized walking joint space trajectories with a non-linear optimization approach. Again, we utilize a sum of squares (SoS) approach that uses the trust-region-reflective algorithm (Nocedal and Wright, 2006).

The optimization problem is of the form,

$$\min_{q} \mathcal{J}(\mathbf{q}),\ \mathcal{J} = J^1(q) + \ldots + J^n(q), \tag{5.4}$$

$$subject\ to\ f(\mathbf{q}) - \mathbf{x} = 0, \tag{5.5}$$

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 being a mixture of task and stability constraints. For the purpose of generating quasi-static walking trajectories, our cost function evaluates the following set of cost factors:

- the distance between swing foot position and goal position
- the alignment between swing foot and x versor
- the alignment between swing foot and y versor
- the deviation in pelvis position with respect to the support foot polygon
- the alignment between waist and z versor

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 discretise 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.

We collect 50, full body, joint space trajectories of stepping motions swinging the right leg and the same amount of stepping motions swinging the left leg. Start and goal points of every step are randomized within a reasonable reaching distance.

5.4.2 Implementation

From the 25 DoFs of the Nao robot we separate the 12 DoFs that correspond to the legs and hip joints, while we keep the rest of the joints at a constant pose[1]. We distinguish two phases for each footstep cycle, one for swinging and placing the free leg and a

[1]The position of the hands in this robot does not have a big influence on the stability of the robot as the mass of the hands is small with respect to the mass of the rest of the body. In principle though the hands can be used to provide stabilization while walking.

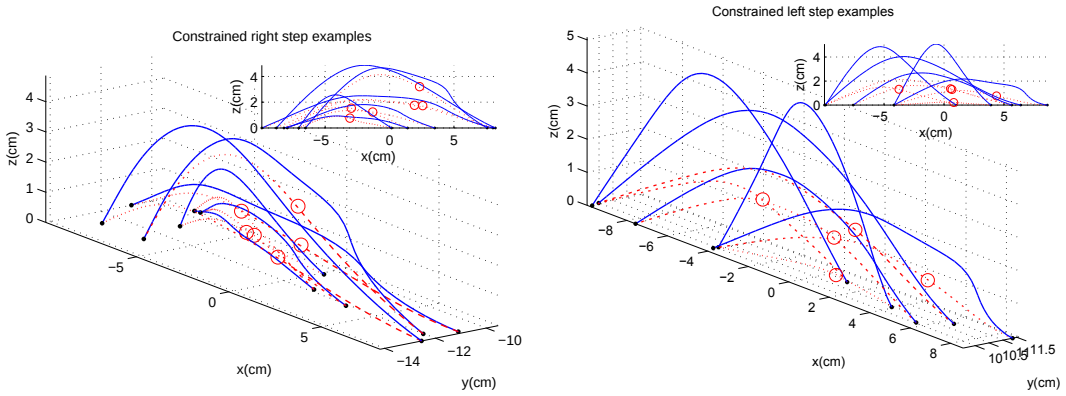


Figure 5.5: Generated constrained task space trajectories from randomly sampled start and end points. The red trajectories correspond to the original unconstrained foot midpoint that collide with the obstacle (red circle). The resulting optimized constrained task space trajectories plotted in blue. Inset 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.)

second for shifting the weight of the robot from the support leg to the swing leg. We treat these two phases as two different skill manifolds as the measure of optimality is essentially different for each phase.

We compute the neighbourhood graph of the demonstration dataset, where we gradually increase the neighbourhood distance threshold until we have a single connected component. A cross-validation step follows that optimizes the model complexity with respect to the decrease in model error. This is done to avoid overfitting the data and boost generalization. Throughout our tests the cross-validation step resulted to 4-dimensional manifolds. All manifold models use 20 RBF kernels and are trained on 500 data points belonging to 25 randomly sampled quasi-static stepping motions, as previously described.

5.4.3 Generation of constrained walking motions

With our framework we are able to produce unconstrained walking trajectories that cover the continuum of the reachable space of the robot's swing leg. In addition, these novel trajectories are produced on average within approximately 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.

To verify the accuracy of the learnt manifold representation we collect a set of randomly sampled walking trajectories and compare against ground truth. The ground truth in this case is trajectories generated through the computational optimization method on the same set of randomly sampled planning queries. 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 in the previous example we randomly pick a set of start and end points in task space and generate an unconstrained stepping trajectory as a geodesic path on the learnt skill manifold. We insert an obstacle near the unconstrained trajectory, keep the starting and goal points of the planning query fixed, and compute the constrained geodesic trajectory. Examples of this process are depicted in Figure 5.5.

These are the task space trajectories of the midpoint of the feet of the stepping Nao, i.e. the generated trajectories through the forward kinematics, while both manifold learning and generation procedures operate in the high dimensional configuration space of the robot. The dashed red lines correspond to the unconstrained predictions that collide with the perceived obstacles, that appear as red circles, while the blue lines are the constrained trajectories that avoid the obstacle points.

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 skilfully avoid the colliding pose and successfully negotiate the step.Snapshots of such behaviour are shown in Figure 5.6 and 5.7.

5.5 Conclusion

We present an approach to constrained trajectory generation over a manifold that compactly encodes a continuous family of trajectories corresponding to a robotic skill.

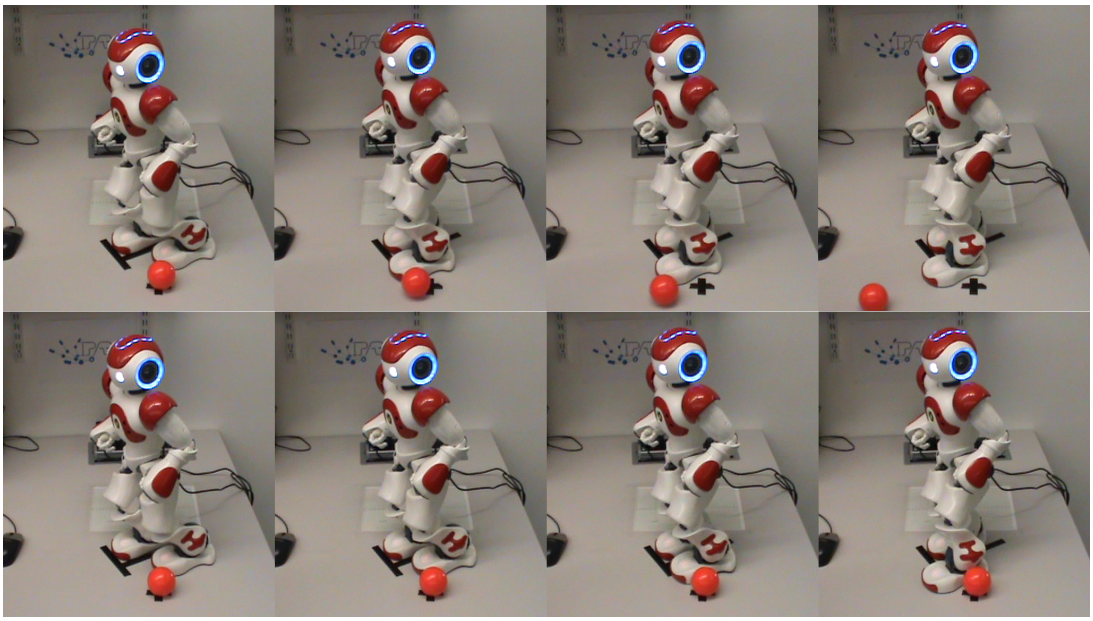


Figure 5.6: Nao executing of a planned motion. *Top row*: the unconstrained stepping trajectory that hits an obstacle (the ball). *Bottom row*: the constrained optimized trajectory where the swing path avoids the obstacle, the ball.

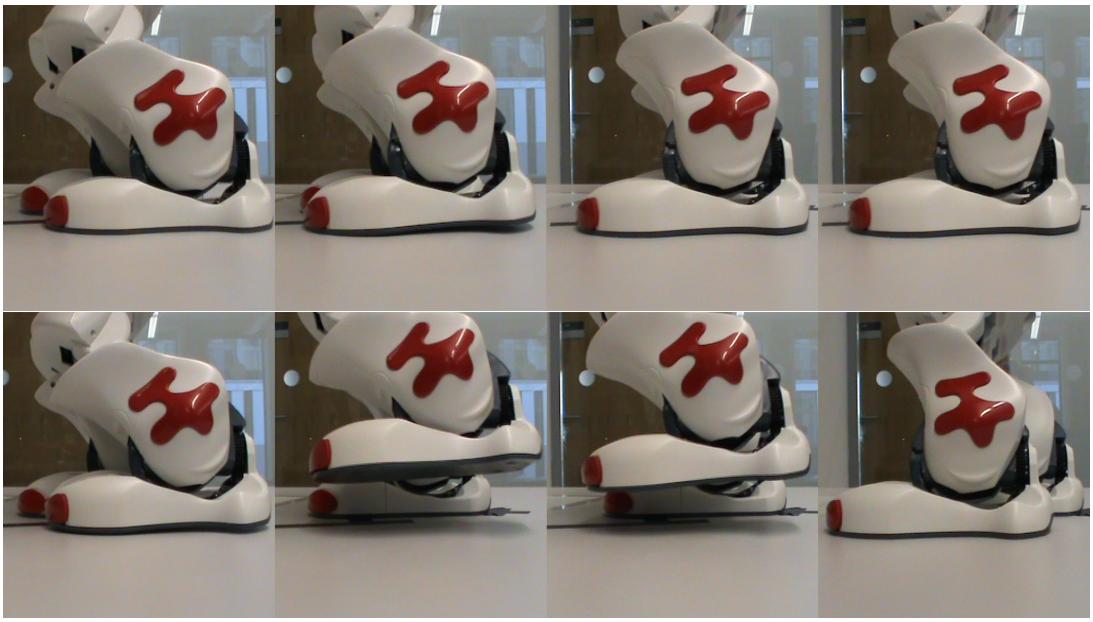


Figure 5.7: Detail of a left foot swing. *Top row*: the original trajectory that provides minimal foot clearance (approx. 2cm on apex). *Bottom row*: adding a obstacle close to the original trajectory pushes the optimization to higher stepping trajectories (here approx. 5cm on apex).

This skill manifold is learnt from demonstration data by approximating the tangent space. Having this manifold along with the local coordinate frames in the form of the tangent space enables efficient ways to handle changing task contexts and obstacles, while respecting intrinsic requirements of the task.

We demonstrated these ideas on two sets of examples - a simulated robotic arm, suitable for visually illustrating core concepts and a humanoid robot behaviour, constrained variable step-length walking. We have demonstrated that the manifolds, defined by solutions to a complex numerical optimization problem, can be learnt from sparse data and that the geometric structure generalizes within and beyond the support of the data. This enables motion synthesis methods where one is able to lazily compute trajectories in response to changing task specifications (perhaps including additions to the underlying cost function expressions) by the constrained geodesics that intrinsically respect the partial specifications that define the essence of the underlying task.

Chapter 6

Reactive control on learnt manifolds

In the preceding chapter we saw how constraints can be added dynamically to a learnt skill manifold, a critical extension for systems that interact with a changing environment. In this chapter we present an approach to reactive control of humanoid robotic systems by utilizing a learnt manifold and a correspondingly derived cost hypersurface, in a model free setting.

The manifold encodes desired evolution of trajectories subject to a variety of specifications including task and system constraints. This structure also enables us to devise a vector field in the ambient space (i.e., the full configuration or joint space) that enforces convergence to the desired family of trajectories from off-manifold points - essential for compensation of large perturbations. This model can be used for efficient trajectory generation and as a metric of closeness to the desired family of solutions. This enables computations such as that of the closest feasible state to any arbitrary state space point. We demonstrate the effectiveness of our approach using robotics examples including a planar 3-link arm, the *Kuka Lightweight Robot* and the Nao humanoid robot.

6.1 The need for strategic control

Many humanoid robotic tasks of common interest require the agent to satisfy skill specifications consisting of constraints including that of the robot's and environmental dynamics. For instance, the task of handling a tray in a typical service scenario requires the robot to achieve one of a certain set of goal states while respecting state constraints that might induce slipping, falling or collisions while also taking care of the manipulator dynamics and internal constraints. Although such a problem is math-

ematically well posed as an optimal control problem, the issue of analytically defining appropriate costs and constraints is a difficult one. In practice, many robotic skills of interest do not admit a clean characterization in such analytical models.

In response to this problem, we have seen the development of approaches that try to induce models and specifications from data - obtained in various ways from people performing the same task. Recent examples include Calinon and Billard (2009), Ijspeert et al. (2002). Typically, these methods are based on the use of supervised learning to approximate models (Gribovskaya et al., 2010; Hersch et al., 2008) or control policies (Schaal et al., 2003; Pardowitz et al., 2007). In almost all such cases, the outcome is a set of trajectories captured in a parameterized probabilistic model that encodes, for instance, possible velocities at each state. This then enables interpolation of trajectories from previously unseen initial conditions.

6.1.1 Control beyond the local model

An issue that has received relatively little attention is that of reactive control in the face of large disturbances. To the extent that the above methods encode possible trajectories, they solve a "local" planning problem regarding the immediate direction along which the trajectory should be executed. One is then expected to combine this with a lower level controller that locally enforces this trajectory. Our interest is in exploring alternate methods that more tightly integrate planning and control. In many applications of interest, we may not have exact dynamics models that are required for the use of traditional control design methods. Indeed, the most common use of data-driven models has been to fill precisely this gap by providing a learnt model that can be plugged into the local controller. In this sense, the type of control that is achieved in many robot learning designs is closely related to corresponding analytical counterparts.

Although there are a number of different techniques for designing controllers to enforce a trajectory, ranging from the simple PID controller to sophisticated $\mathcal{H}_\infty$ designs, the essential objective of all of these controllers is to devise a vector field that guides the system towards a goal - which may be a singleton state, a set of states or a more structured subset or surface. The algorithmic complications of some of the more advanced controller architectures, necessitated by practical issues such as sensorimotor noise or model imprecision, may sometimes obscure this essential requirement but the fact remains that control is a means to identifying the appropriate spaces within which one needs to apply corrective vector fields. Moreover, for the purposes of humanoid

robotics, one may also wish to endow these vector fields with further structure such as more compliant control along certain subspaces and much more tight control along others. The minimum intervention principle (Todorov and Jordan, 2002) hypothesized to explain aspects of human motor control is an example of this.

Our preferred approach to solving this problem involves two components. 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 consider a Nao robot presented with a set of discrete footholds. Assuming that the motion cannot be performed in a dynamic fashion, the task here would be to stand on one leg and swing the free leg to the next foothold. In this case all variations of feasible swinging motions are captured as trajectories along the stepping skill manifold. Deviations from a stable configuration (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.

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 (2010a)) 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.

A related point is that our problem formulation gives a clear meaning to what the limitations of a skill are. For instance, consider the problem of training a robot to perform a dexterous yoga manoeuvre (e.g., standing on one leg while leaning to a side and stretching the other leg to counterbalance). One might begin with a certain conservative set of example movements and approximate a control strategy from this. Then, in 'live' usage, what happens if the robot is asked to perform a much more extreme version of the same manoeuvre? We, the designer, may know that the limits of the

task are defined by a stability boundary but this is far from obvious in the data. The alternative of explicitly specifying this key requirement, using expert insight, may not always be feasible, e.g., as in the *dynamics filtering* approach (Nakamura and Yamane, 2000; Yamane and Nakamura, 2003). In such a scenario, one could approximate the desired effect by restricting trajectories to the 'next-best' movement on the skill manifold, treating cost surfaces away from the manifold (as will be explained in more detail shortly) as a proxy for the task-specific limit.

In contrast it would be easier to provide a number of examples of the behaviour that one would like the agent to exhibit and have the system learn from such demonstrations. The problem that we address in this chapter is how to infer and learn a compact representation of a continuous family of solutions from a set of trajectories that are demonstrated by an expert. Our intuition throughout this thesis is that the demonstrated trajectories belong to a specific subspace, a *manifold*, in the system state space, and are consistently optimized with respect to a specific (set of) cost(s). Our aim then is to accurately learn and represent such a hypersurface, how to plan and replan given such model, how to systematically quantify our confidence on the produced solutions and how this knowledge can be *used online* for control.

6.1.2 Overview of reactive manifold controller

By adopting a machine learning perspective we are able to learn a model directly from demonstrated data. In addition by considering the underlying cost, present in the examples, as a nonlinear, lower dimensional manifold we gain access to a number of benefits. *First*, the learnt manifold provides a systematic way of evaluating distances in state space, it can be viewed as a way of evaluating the state cost, as inferred directly from data. In other words this is a cost metric learnt directly from data, that allows us to produce novel solutions that follow the demonstrations and quantify our confidence on our predictions. *Second*, on the planning phase we can generate paths that generalize the example solutions, i.e., are subject to (an approximation) of the same cost function. *Third*, on a path execution phase, a tracking controller can use this metric to reject perturbations by coupling the distance to the cost manifold with the feedback gains. This way, control can be compliant with respect to perturbations that follow the manifold (cost) hypersurface, but stiff against forces that drive the state of the system away from the model. *Last*, it is unclear how one can reason about the limitations of the task at hand: if you ask the robot to go somewhere that is too far from where

it ought to be, can it generate a safe next-best plan? Our framework shows how one might approach this question. In essence, finding the closest best point would be the same as finding the projection of the state in question onto the manifold.

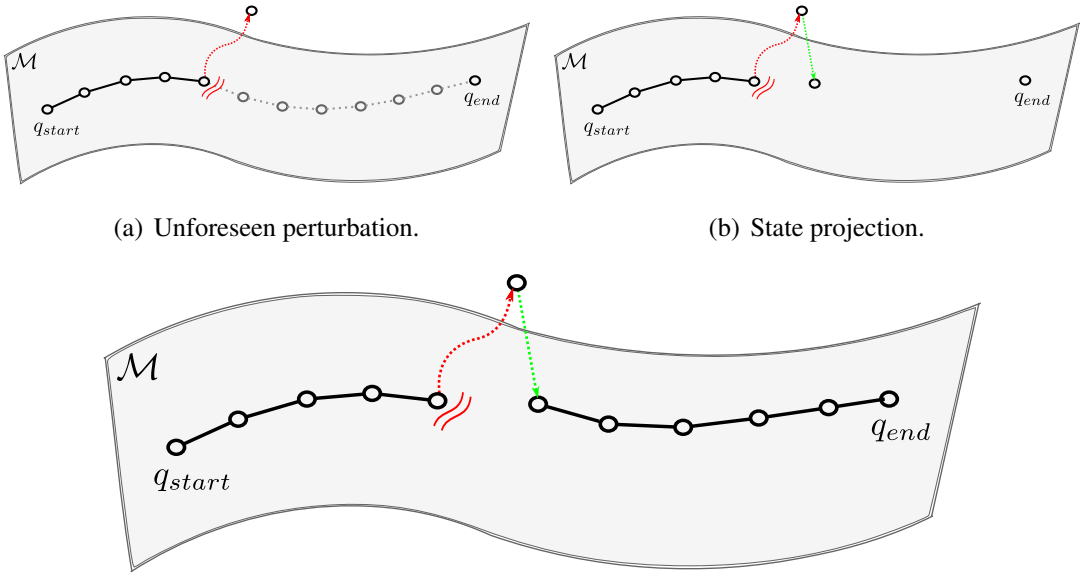
A key contribution here is a method for defining a manifold metric that is used to evaluate the "*value*" of states (in ambient space, e.g., the full joint space) in a general *model-free* fashion. We demonstrate the utility of this approach using three examples. Firstly, we demonstrate efficient path planning, fast re-planning and online control in the face of relatively large scale perturbations for a simulated 3-link arm. Here, both the underlying surface and the learnt manifold can be visualized and intuitively understood. Then, we move on to an anthropomorphic robot arm (the *Kuka* Lightweight Robot arm), using which we demonstrate how our method applies to a realistic task scenario. Last we show how a more complex skill manifold can be used as a bounded planning manifold for the Nao humanoid robot standing on one leg. We begin, in the next section, with an outline of the basic manifold learning method and a detailed presentation of the state projection procedure.

6.2 Control *on* and *to* a skill manifold

Presented with D-dimensional data that is derived from a d-dimensional geometry embedded in the high dimensional space, we can learn a representation of such a geometry with the tools we have developed in the previous chapters. The manifold learning method we have presented in Chapter 3 provides us with a global model of the underlying hypersurface's tangent basis, $\mathcal{H}_\theta$, which serves as a compact encoding of the manifold geometry.

Given such a representation we are then able to produce unconstrained solutions to novel motion planning queries as geodesic trajectories on the learnt manifold. Next we saw how such paths can be modified to accommodate dynamic constraints that may transiently appear in the system's state space, without the need of relearning the representation. The outcome of the above is a series of points in the system's state space, $\mathbf{q} = [q_{start}, q_2, \ldots, q_{end}]$, that follow the manifold's geometry that in turn encodes task and system costs and constraints that where present in the demonstrated solution set while avoiding "no-go" patches that appear in the system's lifetime.

In the preceding chapters we have concentrated our efforts to 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



(a) Unforeseen perturbation.

(b) State projection.

(c) On-manifold replanning.

Figure 6.1: 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.

the manifold geometry is indeed the set of desired states. Such a measure of "value" can be difficult to be explicitly encoded in terms of an analytical cost function but 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.

It is often the case that perturbations will cause the state to leave the manifold geometry (Figure 6.1(a)). Replanning from an off-manifold state would then be insensible 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 6.1(b)). Once the state has returned to the manifold hypersurface, replanning occurs normally

(Figure 6.1(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. As we show in the following sections this is a crucial property for control and stability in the classing sense. This is archived via a projection operation as detailed in the following subsection.

6.2.1 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. Our aim to find 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 geometry.

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

$$d = \left\| q - q' \right\|_2^2.$$

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.

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. Pseudocode for the procedure appears in Algorithm 4.

6.3 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 approxi-

Algorithm 4 On-manifold state projection

INPUT: $\mathcal{M}$, q_{init}, q

OUTPUT: q'

$q' = q_{init}$

$d = \infty$

$H' = orth(\mathcal{M}, q')$ {Tangent basis}

while $d \neq d'$ **do**

$d' = d$

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

$H' = orth(\mathcal{M}, q')$

$d' = \sqrt{(q - q')^2}$

end while

mately 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 two types of corrections may be handled differently and with different levels of stiffness.

Figure 6.2 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. 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.

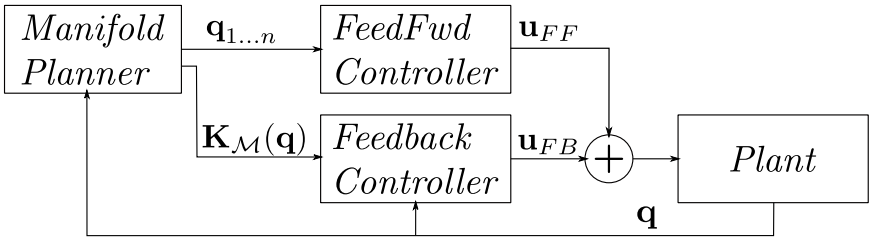
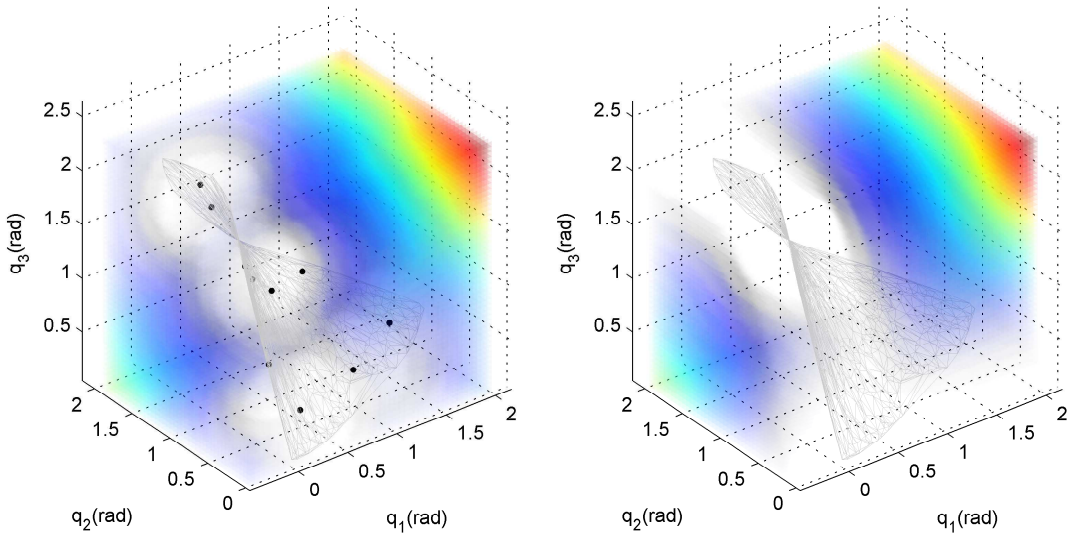


Figure 6.2: A diagrammatic outline of the proposed control strategy utilizing the learnt manifold.

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 centres of clusters of demonstrated data points. Stated in terms of our model, this is similar to defining errors in terms of the centres of RBFs used in our approximation. However, such a metric would yield significantly 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 6.3(a). 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 centres 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 *modelled subspace* directly we can get an accurate and smooth metric, that captures the distance accurately as shown in Figure



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

Figure 6.3: *a*) Volumetric plot of the distance metric that can be evaluated directly from the model. The metric breaks down as the distances get smaller, leading to overestimation of the true distance. Black dots represent RBF centres, over which the evaluation is based. *b*) Volumetric plot of metric derived from our model. The metric evaluates the distance to the *modelled* 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.

6.3(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.

6.4 Manifold control on the 3-link arm

The intent of this example is to elucidate basic concepts underlying reactive control with skill manifolds. We use a *3-link planar* arm where we can explicitly visualize both the configuration space and the optimization manifold (corresponding 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).

We randomly sample 100 Cartesian points from the upper semicircle of the task space of the system. 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 6.4(a), as black dots in joint space and task space plots. The thin gray mesh surface that appears in most related figures is produced by densely sampling the manifold and helps convey a clearer perspective of the geometry of the space in question.

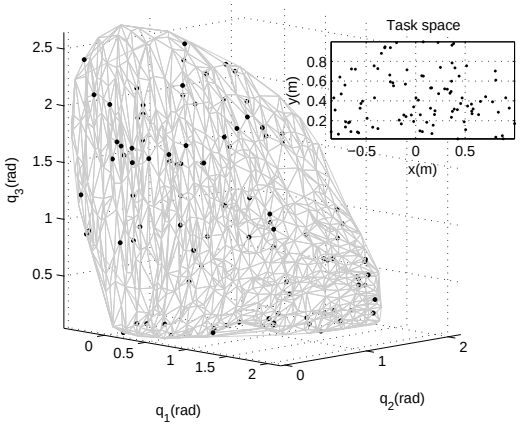
The system being redundant, one needs a redundancy resolution strategy, which implicitly specifies the geometry of the manifold (Figure 6.4(a)) that we subsequently learn. Here, we choose the joint space configuration, $\mathbf{q}$, that minimizes the absolute sum of joint angles, in a different view it minimizes the distance to a convenience (e.g., minimum strain) pose, $\mathbf{q}_c = (0,0,0)$, with joint weighting,

$$\min \|w(\mathbf{q} - \mathbf{q_c})\|^2, \tag{6.1}$$

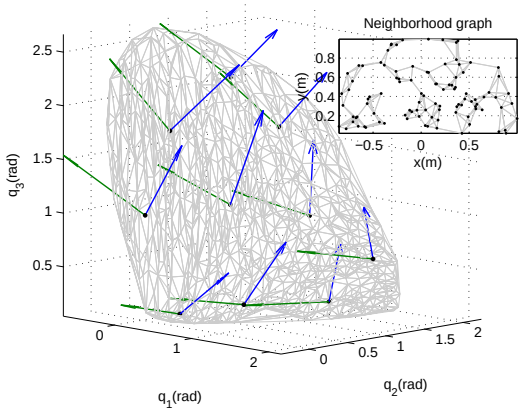
$$\textit{subject to } f(\mathbf{q}) - \mathbf{x} = 0, \tag{6.2}$$

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 its motion.

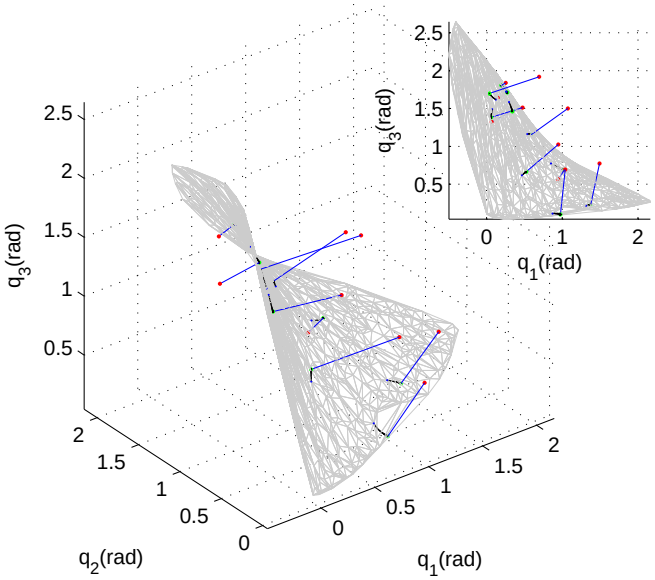
The resulting $\mathbf{q}$'s trace a smooth nonlinear manifold in joint space, depicted in Figure 6.4(a). We note that the manifold surface resembles a convex strip that bends backwards towards the edges, much like a section cut of a bent tube. This is the surface



(a) Samples.

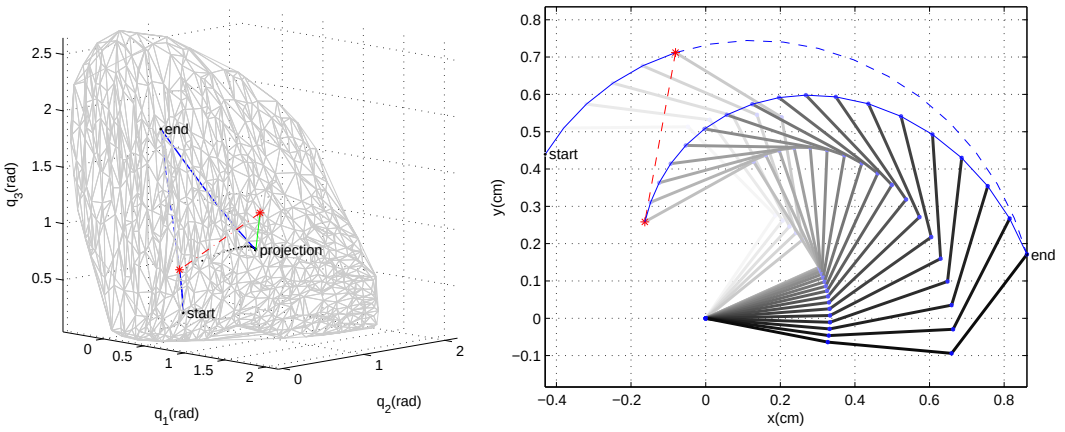


(b) Learnt manifold model.



(c) Random projections.

Figure 6.4: Use of the manifold learning method illustrated for the 3-link arm example. *a*) Start with 100 data points in joint space (the ambient space) that correspond to task space coordinates as in the inset plot. *b*) The neighbourhood graph in task space (inset plot), and the learnt tangent space that the model predicts, shown at the RBF centres in the high dimensional space. *c*) 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) Perturbed geodesic path.

(b) Perturbation example projections.

Figure 6.5: 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.

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 visualisable, for many complex problems.

6.4.1 Implementation

We start by computing the neighborhood graph of the data points. We do this by evaluating task space distances using known forward kinematics. As we require that our set consists of one connected component, we gradually increase the neighborhood distance until no disconnected subsets exist. The resulting neighborhood graph is depicted in Figure 6.4(b)(inset plot).

We can see that the manifold can be naturally represented with a two dimensional tangent space, and we learn a model of $\mathcal{H}_\theta$ with 10 RBFs. We can subsequently evaluate $\mathcal{H}_\theta$ at any point in our joint space. For example Figure 6.4(b) shows the tangent basis

evaluated at the centres 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 traverse 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 6.4(b) for each point in question.

6.4.2 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 6.5). We calculate the average cost per trajectory and average over the results for each case (Table 6.1). The evaluation shows that with the use of manifold 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.

6.5 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 6.6.

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:

$$\min(J), \tag{6.3}$$

$$\textit{subject to } f(\mathbf{q}) - \mathbf{x} = 0, \tag{6.4}$$

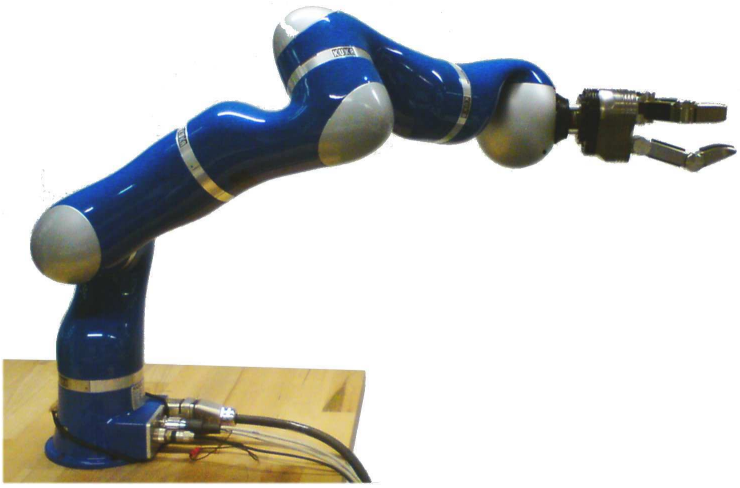


Figure 6.6: The redundant *Kuka Lightweight Arm*.

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

$$J_1 = w\mathbf{q}^2, \tag{6.5}$$

$$J_2 = T^2_{pitch} + T^2_{roll}. \tag{6.6}$$

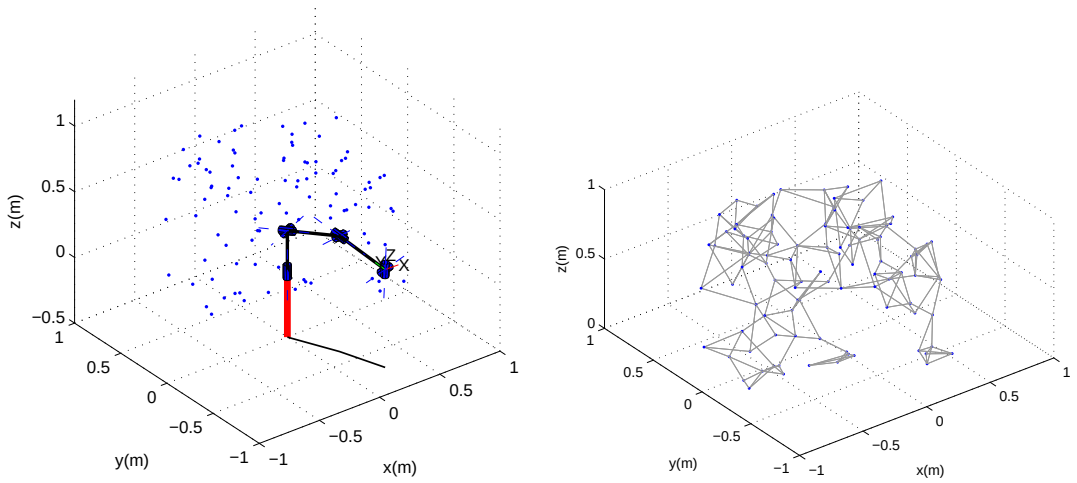
Where T is the end-effector orientation with respect to the global frame of reference. As the system is redundant we use a non-linear optimization method to obtain random training points.

6.5.1 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 6.7(b), 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 6.7(a). 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



(a) Samples. (b) Neighborhood graph.

Figure 6.7: *a*: The kinematic model of the Kuka LightWeight Arm along with 100 randomly sampled joint space endpoint positions. *b*: The neighbourhood graph that results from the sampled joint space points.

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.

6.5.2 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 with a naive method quintic polynomials as in Craig (1989). We then generate a random perturbation at time $t = 0.25$ and replan with both methods (Figure 6.8). We then 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 6.1), 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 (approxi-

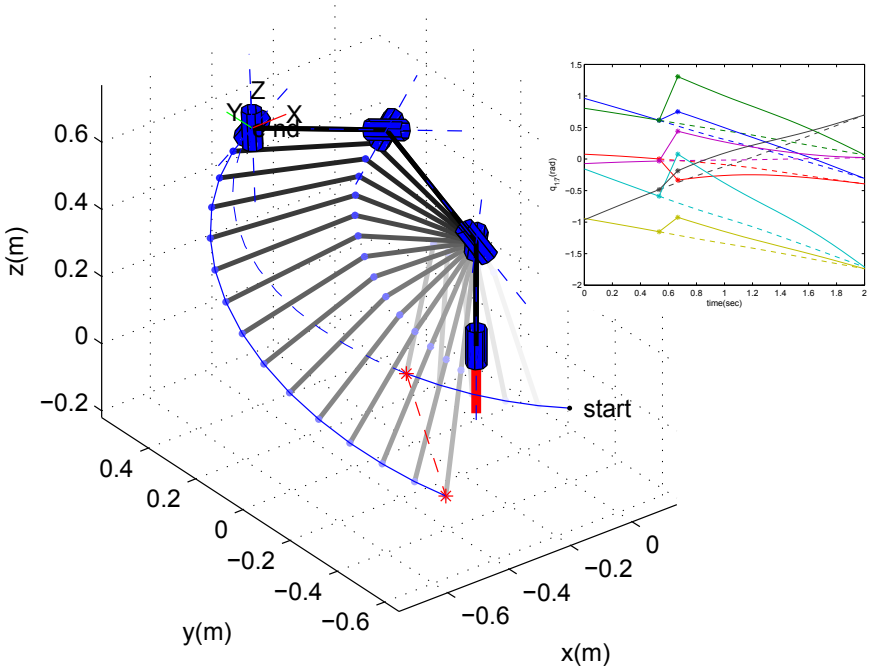


Figure 6.8: An example planned trajectory. The unperturbed trajectory appears as a dashed line. The perturbation pushes the state away from the planned trajectory (red line). The solid blue line, originating at the point when the perturbation ceases (red star), is the replanned trajectory that extends from the projected state point and ends at the goal position. *Inset plot*: Example joint space trajectories including a large perturbation.

mately) optimal cost. The naive method seeks to reach 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 6.1 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 our manifold method provides superior results regarding the underlying cost metrics.

Figure 6.8 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 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

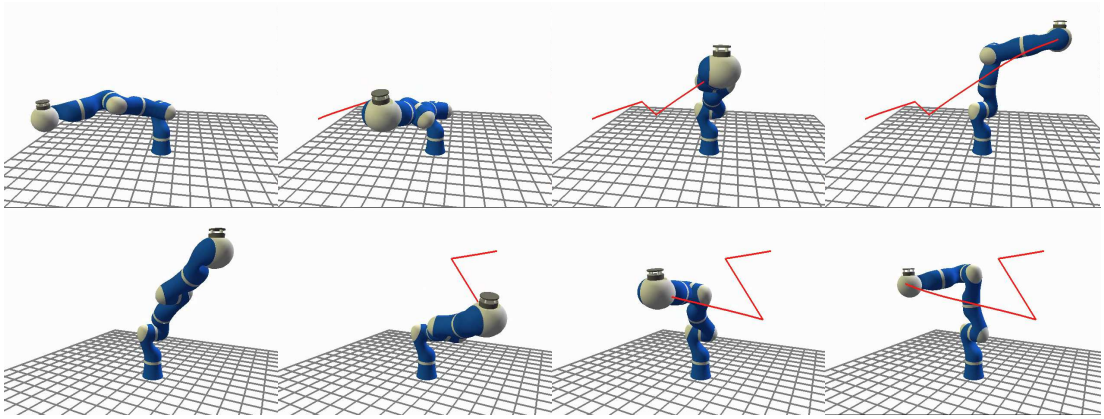


Figure 6.9: Two examples of perturbed trajectories with the Kuka LWR arm. The red line traces the path of the end-effector (the perturbation is seen as the discontinuity in this trace). Time flows from left to right.

closest point that satisfies the learnt cost modelled 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 6.9 contains snapshots from example trajectories on a realistic simulator[1]. 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 we seed our algorithms with 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 would not be in a position to make similarly quantitative statements regarding performance.

[1]The simulator is developed in the *Statistical Learning and Motor Control* group (*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 6.1: 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

6.6 Standing on one leg with the Nao humanoid

The following experimental setup presents a realistic task scenario on the Nao humanoid robot. It aims to provide a concrete example of both the efficiency and efficacy of our approach, as well as to demonstrate in an intuitive setup the concept of skill boundary definition - in this example, the union of the states that are stable.

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 core idea of this section.

We consider a more elaborate example that is implemented on the 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.

6.6.1 Implementation

For this example we use the Nao humanoid robot. The total degrees of freedom of the robot are 25 but for our experiment we only 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;

$$\mathbf{x} = \begin{cases} x \in [-20, 20] \\ y \in [-20, -5] \\ z \in [0, 15] \end{cases} \tag{6.7}$$

This covers a large volume of the reachable set of the free swing leg (Figure 6.10). Nonetheless this sampling produces points that are not reachable without taking an extra step 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 to a standard constant position.

6.6.1.1 Numerical optimization

Mapping from a 3-dimensional task position to a 12-dimensional pose is a general redundancy resolution problem, which we treat as a constrained nonlinear optimization problem. Algorithmically, we use a sum of squares (SoS) approach (Nocedal and Wright, 2006). The optimization problem is of the form,

$$\min_{q} \mathcal{J}(\mathbf{q}), \tag{6.8}$$

$$\mathcal{J} = J^1(q) + \ldots + J^n(q), \tag{6.9}$$

$$\textit{subject to } f(\mathbf{q}) - \mathbf{x} = 0, \tag{6.10}$$

where $\mathcal{J}$ is the cost function that is composed of a number of cost factors J^n, $f(\mathbf{q})$ is the forward kinematics mapping of the pose $\mathbf{q}$, and $\mathbf{x}$ is the sampled task space positions. The search space q is also subject to inequality constraints that keep all

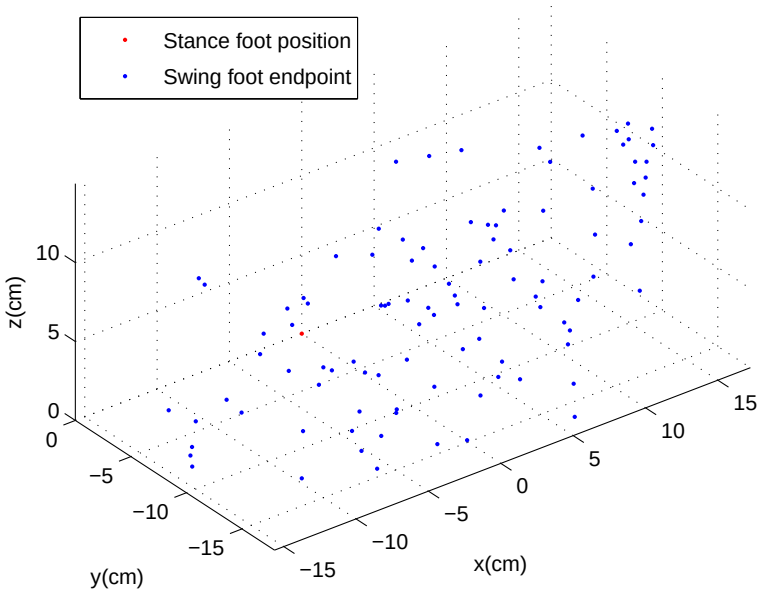


Figure 6.10: Task space samples drawn from a standard uniform distribution on the interval in 6.7. Unreachable samples are discarded through the numerical optimization step. The red dot signifies the position of the stance foot while the blue dots represent the midpoint of the foot that moves freely.

produced configurations within the joint limits of the humanoid. The cost function is a mixture of task constraints and stability constraints. In detail, the cost factors that we used for data generation evaluate:

- 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. 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.

6.6.2 Evaluation

Table 6.2: 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.

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 would it 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[2]. 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

[2]Porting all code to *mex* files would in principle provide a great speed-up but was not investigated.

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 as the evaluation of which is beyond the scope of this thesis. An overview of aforementioned evaluations is available in Table 6.2.

Figure 6.11 and Figure 6.12 provide an example of state projection with the Nao humanoid. Random poses are generated in the systems state space. All these are unstable poses that 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 belongs to 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 6.11 while Figure 6.12 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 chapters. 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.

6.7 Conclusion

Humanoid robotics researchers are increasingly beginning to take note of the need for integrated planning and control schemes that accommodate sophisticated specifications arising at all levels from joint-level limits to global stability and other multivariate constraints, e.g., Berenson et al. (2009b), Ramamoorthy and Kuipers (2008).

In this chapter, we presented an approach to solving this problem by utilizing a learnt manifold and a correspondingly derived cost hypersurface, in a model free setting. Here, we assume that demonstrated samples are generated from a number of different goals (representing task variation) under a consistent cost specification. With this, we show how our method is able to accurately capture this underlying cost as a hypersurface in the ambient high-dimensional space.

We demonstrated how this model can be used for efficient trajectory generation and control. We show how the distance to the learnt manifold can be viewed as a

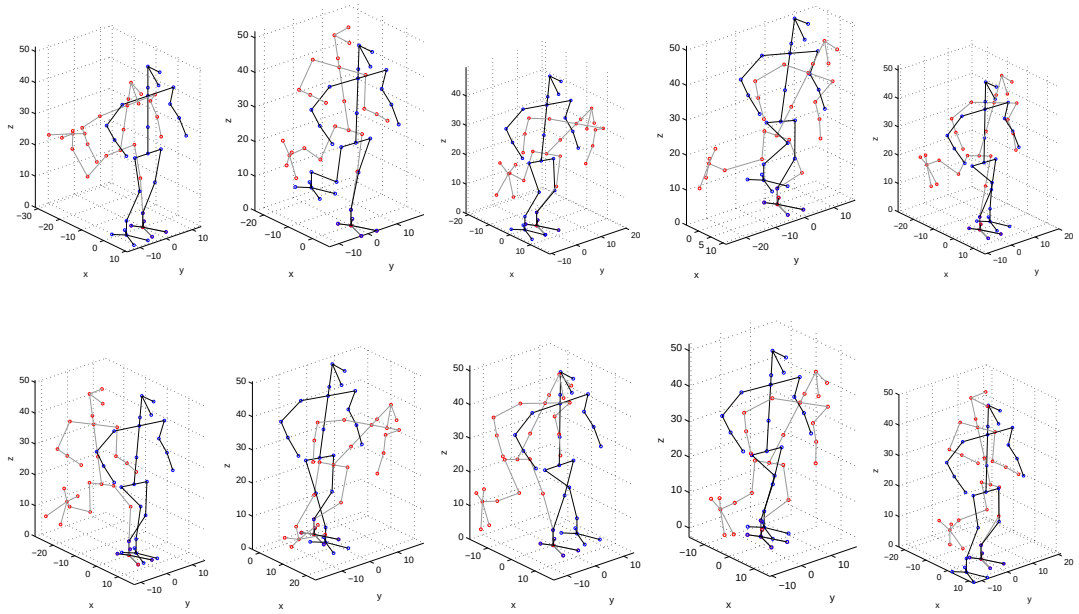


Figure 6.11: The skeleton model of the Nao humanoid performing a number of poses that balance on one leg. The gray skeleton represents the random pose sample that has been sampled within the space of kinematic constraints of the system. The black skeleton is the projection of the previously randomly sampled pose on the stable configuration manifold. The snapshots on Figure 6.12 correspond to the same sequence of poses.

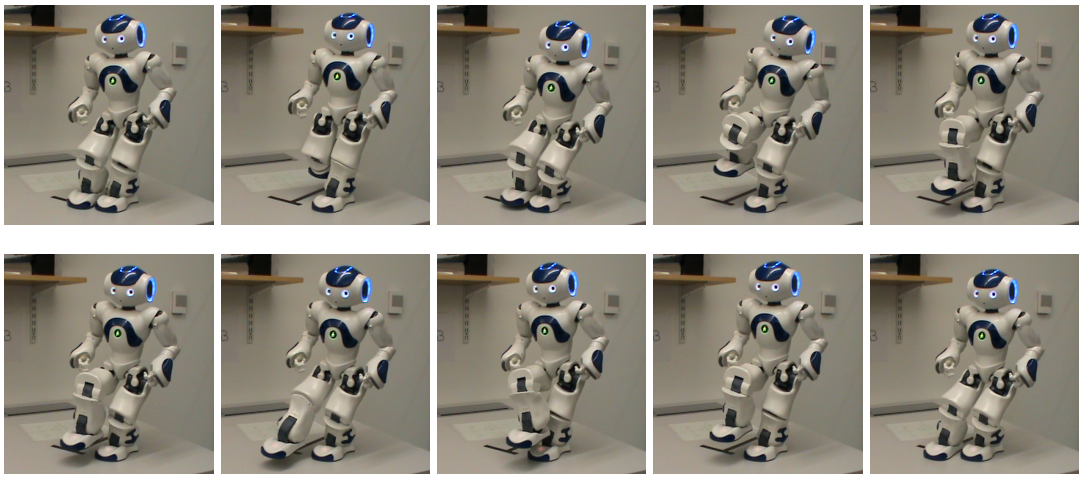


Figure 6.12: Snapshots of the video that demonstrates how the Nao humanoid can move through the poses presented in Figure 6.11. The motions from one pose to the next is a geodesic path on the stable configuration manifold.

metric of closeness to the desired *family* of solutions, while being able to directly compute the best feasible state given an arbitrary ambient state space point. This opens many avenues for future work, including the possibility of devising advanced planning methods such as *Anytime Repairing A** (*ARA**).

The next chapter evaluates our framework against a state of the art imitation learning method.

Chapter 7

Evaluation

In the preceding chapters 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 chapter 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 retargeting, 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.

7.1 Learning by demonstration

Learning by Demonstration (LbD) is one of the two state of the art *Imitation Learning* approaches, the other being *Dynamic Motion Primitives* (DMP). 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. We preferred to compare our framework against LbD, instead of DMP, as our approach is more similar to LbD than DMP. This similarity will become

more clear as we discuss the LbD framework in the following subsection.

7.1.1 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 (Calinon and Billard, 2008, 2009).

The dynamics of the data generating system, f, are approximated from a given set of demonstrated trajectories. This produces an estimate, $\hat{f}$, as a nonlinear combination of a finite set of Gaussian kernels that define a joint probability distribution function (PDF), $P(\xi^i, \hat{\xi}^i)$, over a training set of example trajectories $\xi^i, \hat{\xi}^i, i = 1, \ldots, M$. The PDF is defined as a mixture of K Gaussian $G^1, \ldots, G^K$, with μ^K and Σ^K being the mean and covariance matrix of a Gaussian G^K:

$$P(\xi^i, \hat{\xi}^i) \;=\; \frac{1}{K} \sum_{k=1}^{K} G^K(\xi^i, \hat{\xi}^i; \mu^k, \Sigma^k), \tag{7.1}$$

and

$$\mu^k \;=\; [\mu^k_{\xi}; \mu^k_{\dot{\xi}}] \;\; and \;\; \Sigma^k \;=\; \begin{pmatrix} \Sigma^k_{\xi} & \Sigma^k_{\xi\dot{\xi}} \\ \Sigma^k_{\dot{\xi}\xi} & \Sigma^k_{\dot{\xi}} \end{pmatrix}, \tag{7.2}$$

where each Gaussian probability distribution G^k is given by:

$$G^K(\xi^i, \hat{\xi}^i; \mu^k, \Sigma^k) = \frac{1}{\sqrt{(2\pi)^{2d}|\Sigma^k|}} e^{-\frac{1}{2}(([\xi^i_t, \dot{\xi}^i_t] - \mu^k) T (\Sigma^k)^{-1} ([\xi^i_t, \dot{\xi}^i_t] - \mu^k))}. \tag{7.3}$$

The model is trained with EM and generation of a new trajectory from a learned GMM is done by sampling equation 7.1, the GMR process. GMR produces velocity estimates of the state vector of the system, the velocities are then integrated to the state and a new state vector is produced, in turn feeding into the GMR process. The procedure terminates either after a finite number of iterations or when the state has converged to an attractor point in the system's state space.

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.

7.2 iCub data

The dataset used for the comparison of the two methods comes from the iCub humanoid robot[1]. It consists of 5 trajectories, discretised 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.

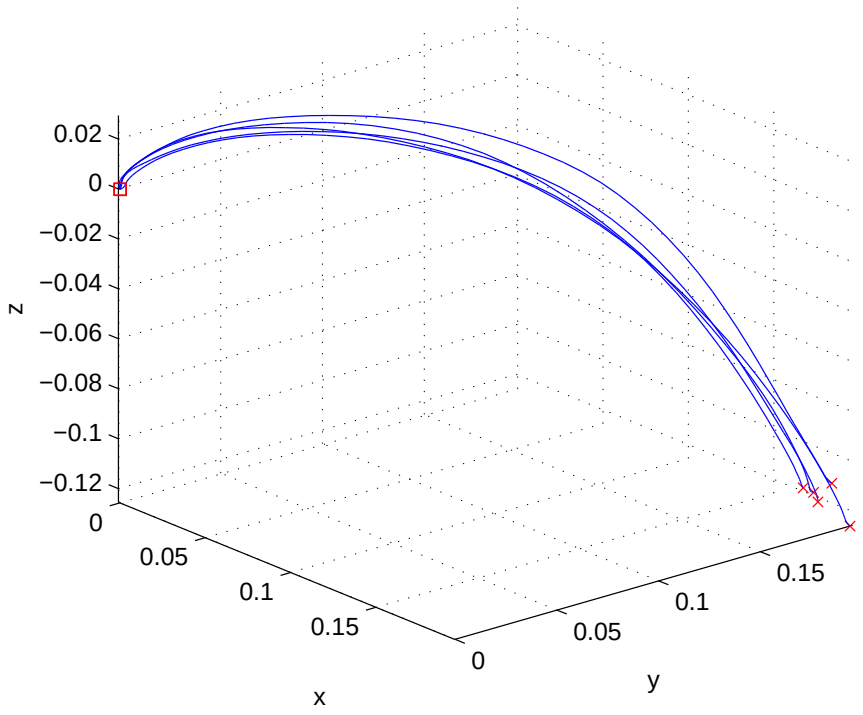


Figure 7.1: iCub *pick-and-place* task. The data consists of 5 trajectories of 100 datapoints each.

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 7.1.

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

7.3 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 requires the training of two GMM/GMR models, one used for learning and predicting positions, and the other 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:

$$\mathbf{x} = [x, y, z, \dot{x}, \dot{y}, \dot{z}]^T. \tag{7.4}$$

Each kernel, $G^{k=1,\ldots,6}$, 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 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 win a *cross-validation* manner, as this is sufficient to explain the dataset at hand.

We begin with the calculation of the nearest neighbourhood 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 neighbourhood graph that results from this operation is available on Figure 7.2. Next we learn a manifold model, $\mathcal{M}$, with the procedure described in section 3.3. The computation of the neighbourhood graph as well as the manifold learning, requires approximately 1.38 seconds to complete.

7.4 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

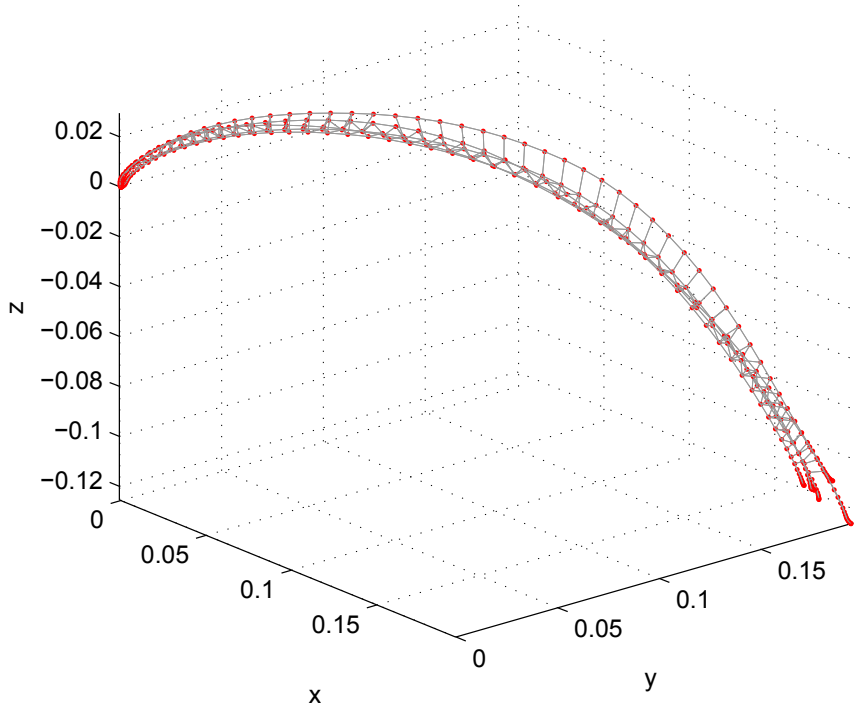


Figure 7.2: The neighborhood graph that results from the iCub data.

Table 7.1: Evaluation of both methods with regards to training and generation times. The results are averaged over multiple trials (20 and 27 respectively).

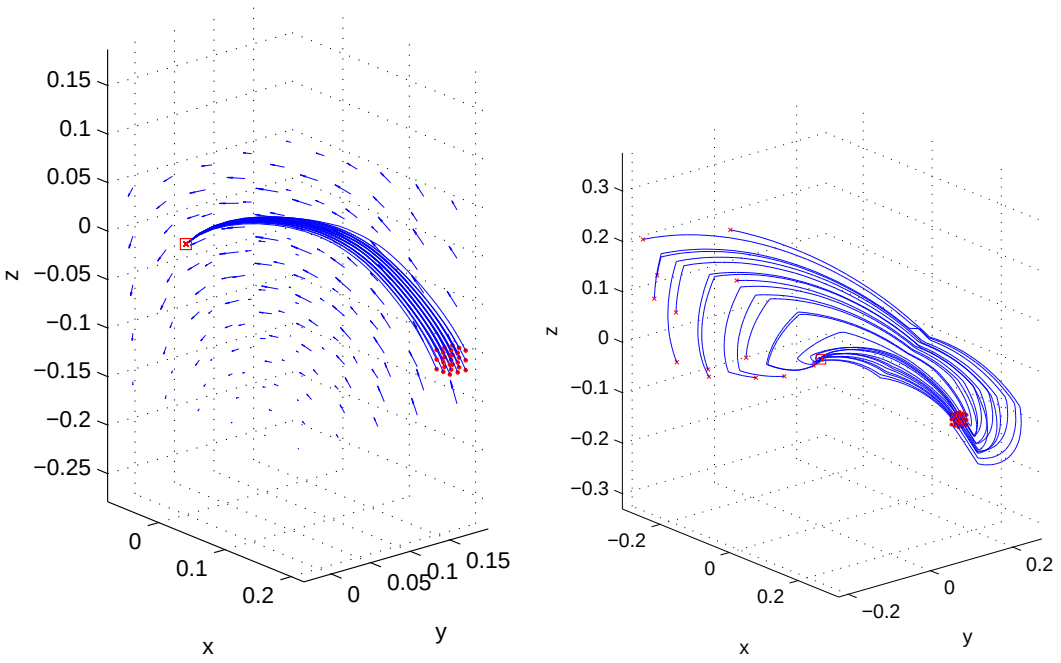
	Number of kernels	*Training time*	*Generation time*
Manifold model	5	$1.38sec$	$0.0817sec$
GMM/GMR	5	$2.35sec$	$0.0853sec$

cube centred at the point chosen in the GMM/GMR 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,

$$\mathbf{x}_{cube} = \begin{cases} x_{init} \pm 0.01 \\ y_{init} \pm 0.01 \\ z_{init} \pm 0.01 \end{cases} (small\ cube),\ \mathbf{x}_{cube} = \begin{cases} x_{init} \pm 0.05 \\ y_{init} \pm 0.05 \\ z_{init} \pm 0.05 \end{cases} (large\ cube). \quad (7.5)$$

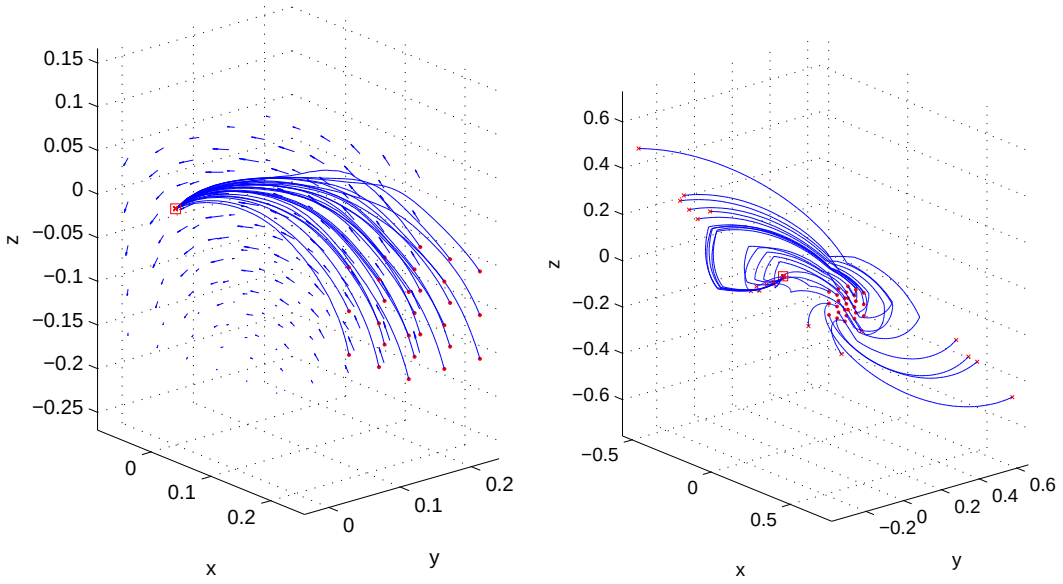
For both cases the target of the trajectories is set to $x_{goal} = [0, 0, 0]$. We run the set of 27 start/end tuples through the GMM/GMR and manifold trajectory generation procedures, both for the small and large cube starting positions. The results of the small cube staring position changes are show in Figure 7.3, while the large cube results are plotted in Figure 7.4. The varying initial positions are marked with red points, the points that each trajectory reaches are red '$\times$' marks, and the goal position is marked with a red square.



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

Figure 7.3: 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.

The outcomes of this process are plotted in Figure 7.3 and Figure 7.4. 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



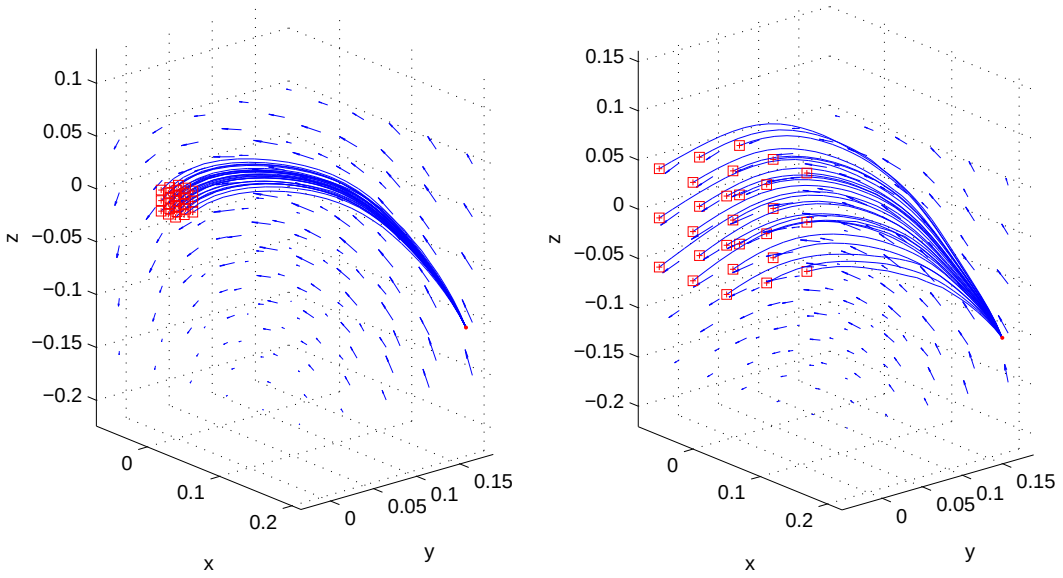
(a) Manifold. (b) GMM/GMR.

Figure 7.4: 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.

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 more strong when we alter the initial position according to the large cube points (Figure 7.4(b)). In contrast the manifold trajectory generation yields very good results, even when we initialize from the points on the large cube (Figure 7.4(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 thesis, thus we present examples of goal changes only for the manifold model.



(a) Small change.

(b) Large change.

Figure 7.5: 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.

The results on Figure 7.5, for both a small and a large changes of the goal position, 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 7.2.

Table 7.2: 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 in *starting* position	*Small*	27	0	12	15
	Large	27	0	8	19
Change of *goal*	*Small*	27	0	–	–
	Large	27	0	–	–

7.4.1 Manifold metric

It is also interesting to investigate what the manifold metric, that we introduced in Chapter 6, would result to 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 7.6 presents a volumetric plot of the manifold metric. The blue lines are the set of demonstrated trajectories, originating from the red '$\times$' marks 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 7.6 demonstrates how the manifold metric creates a tunnel of desirable states that surround the demonstrated solution set.

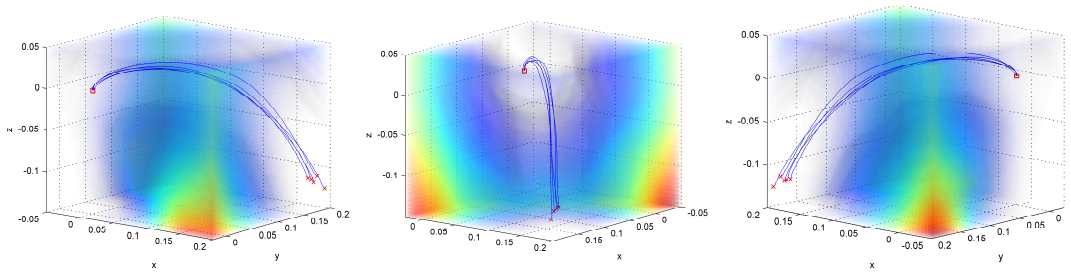


Figure 7.6: Volumetric plot of the manifold metric that allows for reactive feedback control. The color-coding signifies distance to the manifold model, ranging from red (furthest) to blue. Small distance volumes are transparent for clarity. All three figures are rotated views of the same plot and demonstrate how the manifold metric creates a tunnel of desirable states that surround the demonstrated solution set.

7.5 Conclusion

In this chapter we have evaluated the motion planning and control framework that we propose in this dissertation, against a state of the art imitation learning method. We have presented the statistical GMM/GMR LbD framework and we have compared on learning and generation time.

We further compared the two approaches on a realistic dataset generated with the iCub humanoid, performing a *pick-and-place* manipulation task. We have evaluated the two methods with respect to robustness against small and large changes in initial conditions and have demonstrated that our manifold representation significantly outperforms the GMM/GMR method, as the latter is very sensitive to even small changes of starting positions.

Additionally we presented a scenario where the goal of the task changes within a small and a large offset. In this case we showed results from our manifold approach, that exhibits consistently good behaviour. Furthermore, handling such changes is an easy task for the manifold representation while a GMM/GMR model would require a complicated transformation treatment.

Last we demonstrated the form that the manifold metric, presented in the preceding chapter, takes with respect to the solution set at hand. Such a metric of state value, or cost, takes the form of a tunnel around the demonstrated trajectories, and can be used directly in a reactive control manner. This gives our approach the ability to overcome large unforeseen perturbations and stabilize against smaller disturbances.

In the next chapter, we summarize and give conclusions on the work undertaken in

this thesis and suggest directions for future work.

Chapter 8

Conclusions

In this thesis, we have explored an alternate model-free approach to trajectory generation and control that utilizes a manifold learning method as a basis for robotic skill representation. We have presented several examples of learning such skills from demonstrated solution sets, of adding kinodynamic constraints in a dynamic manner and of using the learnt manifold model for reactive control. Furthermore we have presented the utility of our framework in a variety of robotic platforms - ranging from a 3-link planar arm to a small humanoid robot.

In Chapter 2 we reviewed several state of the art methods from planning, control and machine learning literature. In particular we have shown that formal planning and control methods rely heavily on accurate models of the systems in question and models of the interaction of such plants with their environment. Such models are often hard to build and computationally expensive to work with. In addition we showed that trajectory generation methods that utilize some kind of machine learning machinery fail to recognize and exploit the manifold structure that robotic skills invariantly exhibit, while often learning and motion planning/control are carried out in different spaces.

In Chapter 3 we introduced the manifold learning machinery that was used throughout this thesis. We presented a detailed account of the manifold learning algorithm as well as the procedure for learning from a given solution set. We discussed the RBF model used as the base manifold representation, the error that needs to be minimized and the error minimization procedure. Furthermore, we provided an illustrative example of the manifold learning method on a benchmark swiss-roll dataset.

In Chapter 4 we presented how a manifold model can be used for learning a robotic skill and how such a learnt model can generate novel trajectories as geodesic paths. We gave an algorithm for producing such geodesic paths via iterative optimization

of an initial path with respect to the underlying manifold geometry. We evaluated the generalization ability of such novel trajectories while also compared the computational complexity and generation time against the ground truth optimization procedure used to generate the demonstrated solution set. We provided a number of examples, namely on reaching with the 3-link planar arm, and on walking both with the KHR-1HV and Nao humanoid robots.

In Chapter 5 we extended the trajectory generation procedure to account for kinodynamic constraints that are not represented in the training solution set. We demonstrated how such constraints can be later added to the trajectory generation procedure in an on-line manner. We showed how the generative procedure is modified to produce trajectories that avoid patches on the manifold geometry that would result in collisions with the introduced obstacles. We provided a number of examples of unconstrained and constrained trajectories, in a reaching scenario with the 3-link planar arm and a stepping example with the Nao humanoid robot.

In Chapter 6 we demonstrated how the manifold skill representation can be used in a on-line setting for reactive control. We showed how the manifold model gives rise to a natural quantitative measure of the system's state space that can be viewed as a cost and used directly for control. We demonstrated the superior qualities of such a measure in comparison to a naive metric. We provided evidence that such a control strategy results in trajectories that outperform a traditional control method in terms of overall cost as evaluated by the cost function used to provide the ground truth example solutions set. In addition we presented how such a metric can be used to constrain the evolution of trajectories in states that are achieving the task present in the example solution set. We provided a variety of examples that demonstrate how this combined motion planning and reactive control framework can be utilized on a number of example scenarios.

Finally, in Chapter 7 we evaluated our framework against a state of the art imitation learning approach. The evaluation was based on data from the *iCub* humanoid robot performing a pick-and-place task. We demonstrated that our approach provides superior generalization and stability characteristics by investigating the behaviour of both approaches with changes to initial and goal positions. In addition we showed that our manifold based method provides better robustness and far greater ease of retargeting, with respect to changes of both the initial and goal positions.

Future work

There exist a number of directions in which the work presented in this thesis may be extended in future work.

Improved learning machinery

Using an RBF model has been convenient as it is easy to train and simple to work with, both in terms of parametrization and further ability. There are a number of machine learning methods that one can use for learning manifold models. One promising approach would be to use a *Gaussian Processes* (GP) based model. Training such a model may be more complex that training and RBF network but the advantages can be many. Mainly a GP model can provide confidence bounds on its predictions; this way predicted trajectories can be quantitatively classified with respect to skill achievement, a measure of predicted success for each query and a potential cost to be added to each path optimization. In addition, a GP model can be incrementally trained, so that new examples of trajectories can be incorporated to the skill manifold as soon as they become available. This would be more suitable for a robot designed for adaptive everyday interaction as opposed to an one-off learning phase.

Motion recognition

Extending the current framework to allow for motion recognition would be a intuitive forward step. Having a set of learnt skill manifolds, one can easily formulate a classification procedure that in essence evaluates the likelihood that a given example can originate from each of the set of manifolds. Naively this can be quantified by directly computing the distance from each sample to each manifold hypersurface. In principle, measures that take into account the evolution of a trajectory on a manifold should also be considered. This can potentially provide a base for learning from continuous demonstrations, without the need of characterization of each skill set as a single learning trial. For example the robot can compare any new sample with its existing skill library and if the new example does not match any of the existing skills a new skill manifold will be learnt. This also gives rise to the need for incremental learning approaches on the level of single skills and on the level of skill libraries. This way a robot can adaptively improve its skill set throughout its lifetime instead of following a one-shot learning paradigm.

Human data

The experiments that we have presented have all been based on demonstration data that has been generated from an optimization procedure. This has been highly beneficial as we had a clear view of the data generating procedure, thus we were able to quantitative evaluate the success and limitations of our framework. It would be very interesting to have our approach tested with real human demonstrations. Human demonstrations are subject to a variety of factors that make their modelling difficult. The main difficulty has to do with the noise present in human demonstrations. Such noise can manifest in the level of solutions, meaning that for the same planning query we can get a set of similar but different examples. Given imperfect examples, one would need to take into consideration the effect of noise in the learning dataset. On a higher level different people can also employ different strategies for achieving the same goal, leading to examples for the same skill conceptually but of different planning outcomes. In a robotic context the equivalent would be to optimize for the same goal but with a different set of cost factors. Learning from noisy examples along with incremental learning and motion recognition can potentially lead to a learning based framework highly suited for real life learning without the need of expert demonstrations.

Higher order manifolds

Most of the examples in this thesis take place within a robot's configuration space. Points that belong to such space represent the configuration of each degree of freedom of the system. In this case the temporal relationships are captured through the neighbourhood relationship computation step of the manifold learning algorithm, in an indirect manner. An interesting direction of future work would be to extend our approach to higher order spaces such as manifolds of velocities and accelerations. All the machinery that we have developed throughout this work would be applicable to higher order manifolds as well. Such representations can provide further flexibility and grant access to a variety of new optimization criteria with regards to geodesic path generation, e.g. minimum time, minimum jerk, etc.

Basis for Optimal control

As already mentioned in the Chapter 6 of this thesis, a learnt skill manifold can be used in an optimal control setting. One way of doing this would be to include the

learnt manifold as an inequality constraint to the optimization. This way the solutions returned will be trajectories that evolve on the manifold hypersurface. Another place for the use of skill manifolds would be the initialization of optimal control search procedures. Most optimal control methods rely on simplistic initialization methods that often have a high impact on the quality of the result. Learnt manifolds can be used to provide a good initial estimate for the optimization, this way bringing down the optimization time and greatly improving the outcome.

ARA* planner

As briefly mentioned in Chapter 6, the manifold metric that arises naturally through the manifold representation can be used as the basis for more traditional planning machinery. For instance, an interesting research direction would be to use such a leant metric with a planning algorithm such as *Anytime Repairing A** (ARA*). ARA* is a real-time variant of A* and is capable of producing planning solutions of increasing optimality the more computational time is provided. A general purpose planning algorithm such as ARA* can be used to generate the geodesic paths on the learnt manifolds, starting from a crude approximation in computationally restricted scenarios and gradually improving to fine optimality resolution in cases where time criticality relaxes. This can potentially provide a general purpose, flexible realtime planner and controller framework for use in everyday tasks.

Bibliography

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57:469–483.

Arnold, V. I. (1989). *Mathematical Methods of Classical Mechanics (Graduate Texts in Mathematics)*. Springer, 2nd edition.

Athans, M. A. and Falb, P. L. (1966). *Optimal Control*. McGraw-Hill, New York.

Beaudoin, P., van de Panne, M., Poulin, P., and Coros, S. (2008). Motion-motif graphs. In *Symposium on Computer Animation 2008*.

Berenson, D., Chestnutt, J., Srinivasa, S., Kuffner, J., and Kagami, S. (2009a). Pose-constrained whole-body planning using task space region chains. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids '09)*.

Berenson, D. and Srinivasa, S. (2010). Probabilistically complete planning with end-effector pose constraints. In *IEEE International Conference on Robotics and Automation (ICRA '10)*.

Berenson, D., Srinivasa, S., Ferguson, D., and Kuffner, J. (2009b). Manipulation planning on constraint manifolds. In *IEEE International Conference on Robotics and Automation (ICRA '09)*.

Berenson, D., Srinivasa, S., and Kuffner, J. (2011). Task space regions: A framework for pose-constrained manipulation planning. *International Journal of Robotics Research*.

Billard, A., Calinon, S., a. D. R., and Schaal, S. (2007). *Handbook of Robotics*, chapter 59 - Robot programming by demonstration. MIT Press.

Billard, A., Calinon, S., Dillmann, R., and Schaal, S. (2008). Robot programming by demonstration. In Siciliano, B. and Khatib, O., editors, *Handbook of Robotics*, pages 1371–1394. Springer, Secaucus, NJ, USA.

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

Bitzer, S., Havoutis, I., and Vijayakumar, S. (2008). Synthesising novel movements through latent space modulation of scalable control policies. In *Lecture Notes in Computer Science*, pages 199–209. Springer Berlin / Heidelberg.

Boor, V., Overmars, M., and van der Stappen, A. (1999). The gaussian sampling strategy for probabilistic roadmap planners. *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, 2:1018–1023 vol.2.

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

Bretl, T., Lall, S., Latombe, J.-C., and Rock, S. (2004). Multi-step motion planning for free-climbing robots. In *in Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 1–16.

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

Calinon, S. and Billard, A. (2008). 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.

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

Chalodhorn, R., Grimes, D., Maganis, G., Rao, R., and Asada, M. (2006). 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)*, pages 3693–3698.

Chestnutt, J., Lau, M., Cheung, K. M., Kuffner, J., Hodgins, J. K., and Kanade, T. (2005). Footstep planning for the honda asimo humanoid. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G. A., Burgard, W., Kavraki, L. E., and Thrun, S. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA.

Coates, A., Abbeel, P., and Ng, A. Y. (2008). Learning for control from multiple demonstrations. In *Proceedings of the 25th International Conference on Machine Learning (ICML '08)*, pages 144–151, New York, NY, USA. ACM.

Conner, D., Choset, H., and Rizzi, A. (2009). Flow-through policies for hybrid controller synthesis applied to fully actuated systems. *IEEE Transactions on Robotics*, 25(1):136 –146.

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

Conner, D. C., Choset, H., and Rizzi, A. (2006). 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. MIT Press.

Conner, D. C., Rizzi, A., and Choset, H. (2003). Composition of local potential functions for global robot control and navigation. In *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, volume 4, pages 3546– 3551. IEEE.

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

Degallier, S., Santos, C., Righetti, L., and Ijspeert, A. (2006). Movement generation using dynamical systems : a humanoid robot performing a drumming task. In *6th IEEE-RAS International Conference on Humanoid Robots*, pages 512 –517.

Dever, C., Mettler, B., Feron, E., Popovic', J., and Mcconley, M. (2004). Trajectory interpolation for parametrized maneuvering and flexible motion planning of autonomous vehicles. *AIAA Guidance, Navigation, and Control Conference*.

Dever, C., Mettler, B., Feron, E., Popovic', J., and Mcconley, M. (2006). Nonlinear trajectory generation for autonomous vehicles via parameterized maneuver classes. *Journal of Guidance, Control and Dynamics*, 29:289–302.

Diankov, R., Ratliff, N., Ferguson, D., Srinivasa, S., and Kuffner, J. (2008). Bispace planning: Concurrent multi-space exploration. In *Robotics: Science and Systems*.

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

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

Flash, T. and Hogan, N. (1985). The coordination of arm movements: an experimentally confirmed mathematical model. *Journal of Neuroscience*, 5(7):1688–1703.

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

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

Full, R. and Koditschek, D. (1999). Templates and anchors: neuromechanical hypotheses of legged locomotion on land. *Journal of Experimental Biology*, 202(23):3325–3332.

Gribovskaya, E., Zadeh, K., Mohammad, S., and Billard, A. (2010). Learning Nonlinear Multivariate Dynamics of Motion in Robotic Manipulators. *International Journal of Robotics Research*.

Hastie, T., Tibshirani, R., and Friedman, J. H. (2001). *The Elements of Statistical Learning*. Springer.

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

Havoutis, I. and Ramamoorthy, S. (2010b). Geodesic trajectory generation on learnt skill manifolds. *International Conference on Robotics and Automation (ICRA), 2010. Proceedings 2010 IEEE.*

Havoutis, I. and Ramamoorthy, S. (2010c). Motion synthesis through randomized exploration on submanifolds in configuration space. In Baltes, J., Lagoudakis, M., Naruse, T., and Shiry, S., editors, *RoboCup 2009*, volume 5949 of *Lecture Notes in Artificial Intelligence*, pages 92–103. Springer.

Hersch, M., Guenter, F., Calinon, S., and Billard, A. (2008). Dynamical system modulation for robot learning via kinesthetic demonstrations. *IEEE Transactions on Robotics*, 24(6):1463 –1467.

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

Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2003). Learning attractor landscapes for learning motor primitives. *Science*, 15:15471554.

Isto, P. and Saha, M. (2006). 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.

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

Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., and Schaal, S. (2011). Stomp: stochastic trajectory optimization for motion planning. In *IEEE international conference on Robotics and Automation (ICRA '11)*.

Kobilarov, M. B. and Marsden, J. E. (2011). Discrete geometric optimal control on lie groups. *IEEE Transactions on Robotics*, 27(4):641 –655.

Kohonen, T. (1997). *Self-Organizating Maps*. Springer-Verlag, New York.

Kuffner, J. and LaValle, S. (2000). 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.

Kuffner, J. J., Kagami, S., Nishiwaki, K., Inaba, M., and Inoue, H. (2002). Dynamically-stable motion planning for humanoid robots. *Auton. Robots*, 12(1):105–118.

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

LaValle, S. M. and Kuffner, J. J. (2001). Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400.

Levine, W. S. (1996). *The Control Handbook*. IEEE PRESS.

Lewis, A. D. (2007). Is it worth learning differential geometric methods for modeling and control of mechanical systems? *Robotica*, 25:765–777.

Li, W. and Todorov, E. (2006). An iterative optimal control and estimation design for nonlinear stochastic system. In *45th IEEE Conference on Decision and Control, 2006*, pages 3242 –3247.

Nakamura, Y. and Yamane, K. (2000). Interactive motion generation of humanoid robots via dynamics filter. In *Proceedings of the 1st IEEE-RAS International Conference on Humanoid Robots (Humanoids '00)*.

Nakanishi, J. (2004). Learning from demonstration and adaptation of biped locomotion. *Robotics and Autonomous Systems*, 47(2-3):79–91.

Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. Springer, 2nd edition.

Novikov, S. P. and Taimanov, I. A. (2006). *Modern Geometric Structures and Fields*, volume 71. AMS Graduate Studies in Mathematics, 2nd edition.

Pardowitz, M., Knoop, S., Dillmann, R., and Zollner, R. (2007). Incremental learning of tasks from user demonstrations, past experiences, and vocal comments. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(2):322 –332.

Pastor, P., Hoffmann, H., Asfour, T., and Schaal, S. (2009). Learning and generalization of motor skills by learning from demonstration. In *International Conference on Robotics and Automation (ICRA '09)*.

Pontryagin, L. S. (1962). *The Mathematical Theory of Optimal Processes*. CRC Press.

Ramamoorthy, S. and Kuipers, B. J. (2006). Qualitative hybrid control of dynamic bipedal walking. In *Proceedings of Robotics: Science and Systems*, Philadelphia, USA.

Ramamoorthy, S. and Kuipers, B. J. (2008). Trajectory generation for dynamic bipedal walking through qualitative model based manifold learning. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 359–366.

Rasmussen, C. E. and Williams, C. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

Ritter, H. (1997). Self-organizing maps for robot control. In Gerstner, W., Germond, A., Hasler, M., and Nicoud, J.-D., editors, *Artificial Neural Networks ICANN'97*, volume 1327 of *Lecture Notes in Computer Science*, pages 673–684. Springer Berlin / Heidelberg.

Rodriguez, S., Tang, X., Lien, J.-M., and Amato, N. (2006). An obstacle-based rapidly-exploring random tree. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '06)*, pages 895–900.

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

Safonova, A., Hodgins, J. K., and Pollard, N. S. (2004). Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions in Graphics*, 23(3):514–521.

Schaal, S. (2006). Dynamic movement primitives - A framework for motor control in humans and humanoid robotics. *Adaptive Motion of Animals and Machines*, page 261280.

Schaal, S., Ijspeert, A., and Billard, A. (2003). Computational approaches to motor learning by imitation. *Philosophical Transactions: Biological Sciences*, 358(1431):537–547.

Siciliano, B. and Khatib, O., editors (2008). *Springer Handbook of Robotics*. Springer.

Sontag, E. D. (1998). *Mathematical Control Theory: Deterministic Finite-Dimensional Systems*. Springer, New York, 2nd edition.

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

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

Teh, Y. W. and Roweis, S. (2003). Automatic alignment of local representations. In *Advances in Neural Information Processing Systems 15*, pages 841–848. MIT Press.

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

Theodorou E., Stulp F., B. J. and S., S. (2011). An iterative path integral stochastic optimal control approach for learning robotic tasks. In *The 18th world congress of the International Federation of Automatic Control (IFAC' 11)*.

Thomas, S., Morales, M., Tang, X., and Amato, N. (2007). Biasing samplers to improve motion planning performance. *IEEE International Conference on Robotics and Automation (ICRA '07)*, pages 1625–1630.

Todorov, E. and Jordan, M. I. (2002). A minimal intervention principle for coordinated movement. In *In*, pages 27–34. MIT Press.

Todorov, E. and Li, W. (2005). 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*.

Uno, Y., Kawato, M., and Suzuki, R. (1989). Formation and control of optimal trajectory in human multijoint arm movement. *Biological Cybernetics*, 61:89–101.

Verbeek, J. (2006). Learning nonlinear image manifolds by global alignment of local linear models. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 28(8):1236–1250.

Vijayakumar, S., D'Souza, A., and Schaal, S. (2005). Incremental online learning in high dimensions. *Neural Compututation*, 17(12):2602–2634.

Walter, J. and Ritter, H. (1995). Local PSOMs and chebyshev PSOMs improving the parametrised self-organizing maps. In *International Conference on Artificial Neural Networks (ICANN '95), Paris*, pages 95–102.

Wang, J. M., Fleet, D. J., and Hertzmann, A. (2008). Gaussian process dynamical models for human motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):283–298.

Wolpert, D. M. and Harris, C. M. (1998). Signal dependent noise determines motor planning. *Nature*, 394(6695):780–784.

Yamane, K. and Nakamura, Y. (2003). Dynamics filter - concept and implementation of online motion generator for human figures. *IEEE Transactions on Robotics and Automation*, 19(3):421 – 432.

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