Learning and exploiting low dimensional structure for efficient holonomic motion planning in high dimensional spaces

Paul Vernaza *[1] and Daniel D. Lee[2]

[1]The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213
[2]GRASP Laboratory, University of Pennsylvania, Philadelphia, PA 19104

Abstract

We present a class of methods for optimal holonomic planning in high dimensional spaces that automatically learns and leverages low dimensional structure to efficiently find high quality solutions. These methods are founded on the principle that problems possessing such structure are inherently simple to solve. This is demonstrated by presenting algorithms to solve these problems in time that scales with the dimension of a salient subspace, as opposed to the scaling with configuration-space dimension that would result from a naive approach. For generic problems possessing only approximate low dimensional structure, we give iterative algorithms that are guaranteed convergence to local optima while making non-local path adjustments to escape poor local minima. We detail the theoretical underpinnings of these methods as well as give simulation and experimental results demonstrating the ability of our approach to efficiently find solutions of a quality exceeding that of known methods, and in problems of high dimensionality.

1 Introduction

We consider in this work the problem of optimal holonomic motion planning in high dimensional spaces; i.e., we wish to develop an algorithm that finds a continuous trajectory of minimal cost connecting two arbitrary points in some space, subject to holonomic constraints. Unfortunately, most general formulations of this problem are computationally intractable—even in the case that we wish only to find a feasible solution [LaValle, 2006]. However, these results do not bar the existence of interesting special cases that might be solved efficiently. The program of the present work is to identify one such case—the case where the cost function possesses low-dimensional structure; give algorithms to solve this case optimally and efficiently, and finally use these ideas as inspiration to devise algorithms for more general problems that may not possess this structure in a strict sense. Although we cannot hope to find globally optimal solutions in the latter case, we will see that taking inspiration from the optimally solvable case allows us to find methods that monotonically converge to local minima (in the grid refinement limit), make non-local path adjustments to escape poor local minima, and exhibit state-of-the-art performance in practice.

Due to the intrinsic hardness of solving this problem in the general case, most related research has focused on attempting to solve just the feasibility problem. This philosophy spawned the important class of sampling-based planning methods [Amato and Wu, 1996, Kavraki et al., 1996, Kuffner and LaValle, 2000, Hsu et al., 1999, Sánchez and Latombe, 2001], which often succeed in finding feasible solutions quickly and reliably, and with minimal implementation difficulty.

Most approaches to motion planning with cost minimization can be divided into three classes: sampling-based methods [Karaman and Frazzoli, 2010, Ferguson and Stentz, 2006, Diankov and Kuffner, 2007, Berenson et al., 2011], methods based on heuristic search [Likhachev et al., 2004, Likhachev and Stentz, 2008,

*The majority of this work was performed while P. Vernaza was affiliated with the GRASP Laboratory at the University of Pennsylvania.

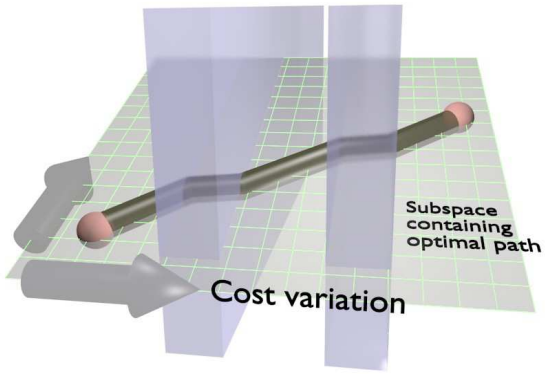


Figure 1: Shown is an optimal path in three dimensions of a cost function that varies only in one dimension. The blue blocks represents high-cost regions, assumed to extend ad infinitum in directions orthogonal to the cost variation direction. The optimal path can be shown to lie in a two-dimensional subspace: the smallest subspace that contains both the direction of cost variation and both of the endpoints.

Vernaza et al., 2009, Cohen et al., 2011], and gradient-based methods [Morimoto et al., 2003, Quinlan and Khatib, 1993, Ratliff et al., 2009]. In this work, we demonstrate a novel method that is founded neither on random sampling of the state space, nor heuristic-based search, nor local optimization. Instead, we first show how problems with a certain kind of low-dimensional cost structure are efficiently solvable exactly (up to discretization). We then show how to extend these ideas to devise practical algorithms applicable to real-world problems, which generally only exhibit approximate symmetries.

This article extends and unifies material previously published in [Vernaza and Lee, 2011a, Vernaza and Lee, 2011b, Vernaza, 2011]. All of the methods and theoretical results presented here were previously published in [Vernaza, 2011], which introduced the results in the context of learning physical systems and subsequently addressed their application to planning problems. This work offers a clearer, more concise, and unified presentation of these concepts in a pure planning setting. Additionally, this article contains extensive experimental results and comparisons to other methods that were previously unpublished.

The next section introduces at a high level the main concepts upon which our approach is based. Section 3 places these ideas in the context of the current literature. Section 4 details the core theoretical results that make our approach plausible. Sections 5 and 6 give practical algorithms implementing these ideas, discussion of their relation to existing methods, and experimental results.

2 Method overview

We first explain at a high level why low dimensional cost structure is a useful concept in the context of motion planning. To that end, the next section describes informally how this structure can be exploited given that it is known a priori. After establishing that low dimensional cost structure is a useful attribute, we discuss how it can be learned automatically from the problem data.

2.1 Exploiting low dimensional cost structure

The concept of low dimensional cost structure is best illustrated by the simple example shown in Fig. 1. Here we consider the problem of finding a minimum-cost path between two points in $\mathbb{R}^3$, illustrated as small spheres. High-cost regions are illustrated as vertical blocks, assumed to extend ad infinitum in directions orthogonal to the vertical direction. The *cost function* is therefore low dimensional in the sense that the cost can be predicted entirely by its projection onto a single vector—in this case, the common surface normal to the wall-like obstacles.

The minimum-cost path is illustrated in the figure as well. We observe that the path does not stray outside a certain two-dimensional affine subspace. This subspace is further observed to be the smallest affine subspace containing both the endpoints and the direction in which the cost varies. In fact, we can

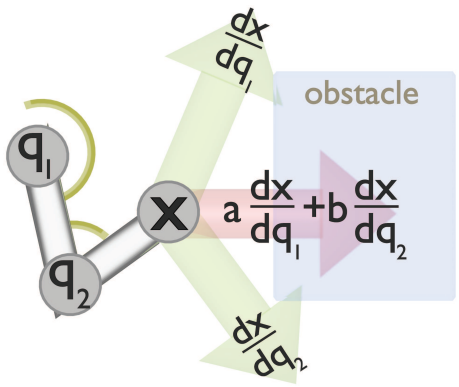


Figure 2: Illustration of how low dimensional structure might arise in a simple arm planning problem. Locally, the cost (obstacle proximity) depends only on a one-dimensional subspace (spanned by $[a\ b]$) of the configuration space determined by the Jacobian (green arrows) and the cost gradient (red arrow).

show that this observation holds generally: the minimum-cost path between two points in a Euclidean space is always contained entirely within the smallest affine subspace containing both points and all directions in which the cost function varies.

The utility of this observation is that, provided all directions in which the cost varies are known a priori, we can find an optimal path by searching over just those paths that lie in a subspace of dimension generally less than that of the entire space. Doing so would yield a modest efficiency gain for the illustrated example, as we could restrict our search to the designated two-dimensional subspace rather than search over the full three-dimensional space. However, the efficiency gain will in general be great, as it is exponential in the amount by which the ambient space dimension exceeds the *cost dimension*, informally defined as the dimension of the set of directions in which the cost varies.

2.1.1 Low dimensional structure in robot arm planning

Fig. 2 shows how this kind of low dimensional structure might naturally arise in the context of arm planning. In this case, we consider a planar arm with just two revolute joints. Suppose we are given a cost function $C(x)$ for this problem proportional to

$$\exp\left(-\min_{o\in\mathcal{O}}\|x-o\|\right),$$

where $\mathcal{O}$ is the set of points inside obstacles, and x is the position of the end effector. The local linearization of the cost function depends only on the gradient $\nabla C(x)$ of the cost—i.e., the direction that locally maximizes the rate at which the end-effector is brought into collision, illustrated as the red arrow protruding into the obstacle orthogonally at its surface. We can express this *principal cost component* in configuration space by the vector

$$\begin{pmatrix} a \\ b \end{pmatrix} := \frac{dx}{dq}^{\dagger} \nabla C(x), \tag{1}$$

where $(dx/dq)^{\dagger}$ denotes the pseudoinverse of the Jacobian that maps configuration-space derivatives to workspace derivatives. This yields a decomposition of the configuration space into a subspace on which the cost depends principally (that spanned by $[a\ b]$) and an orthogonal subspace on which the cost does not depend, under this local approximation.

For arms with more degrees of freedom, we would still hope to be able to make such arguments to motivate the existence of low dimensional cost structure. Unfortunately, as Fig. 2 demonstrates, the particular nature of this structure is dependent both on the geometry of the manipulator and the environment; therefore, we would not expect to be able to deduce so easily the low dimensional structure for an arm with many degrees of freedom in an arbitrary environment.

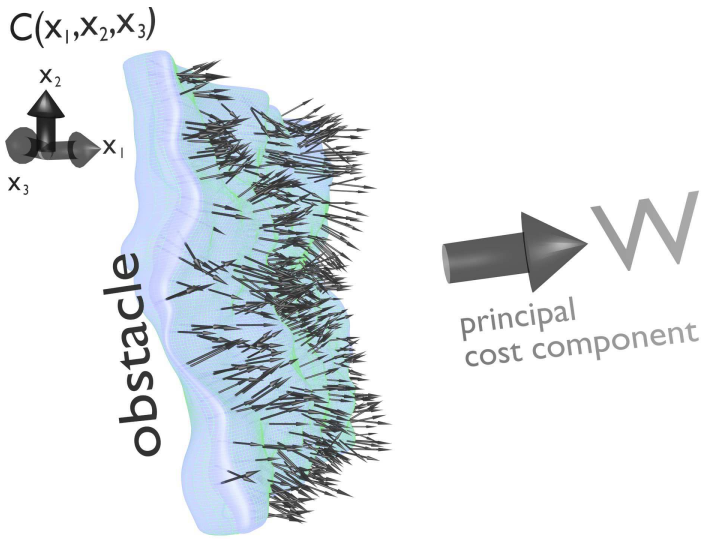


Figure 3: Figure illustrating process of learning low dimensional cost structure by random sampling. Shown is a wall-like cost function that varies in all directions, but exhibiting principal cost variation in the indicated direction. This direction can be discovered approximately by computing moments of sampled cost gradients, illustrated as small arrows protruding from the obstacle, orthogonally to its surface.

2.2 Learning low dimensional cost structure

As in the arm planning problem, it is usually either inconvenient or infeasible to give an a priori characterization of the low dimensional structure of a particular problem. However, it is not difficult to conceive of how it might be learned approximately, as illustrated in Fig. 3. Here we consider the case where the cost is again high in the vicinity of a wall-like obstacle, but the obstacle possesses small undulations such that the cost function varies in every direction, to some extent.

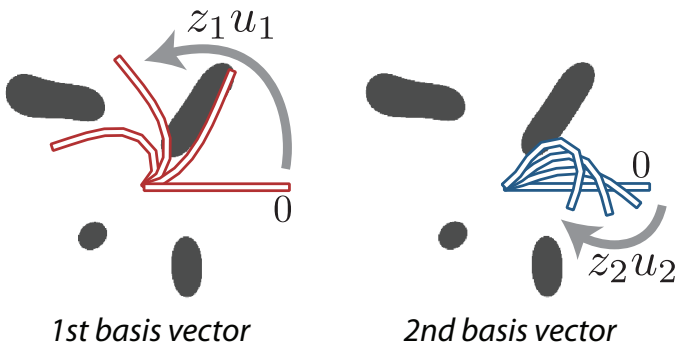
As before, assuming the cost is differentiable, the gradient of the cost indicates the direction in which the cost varies principally at a given point. The low dimensional structure of the cost function as a whole is reflected in the statistics of these pointwise gradients. As such, we define the principal cost component to be the direction in configuration space that maximizes the expected squared directional derivative of the cost function. In order to estimate this direction, we sample gradients of the cost function and perform a simple spectral optimization similar to PCA. Likewise, we find a set of d principal cost components that represent the d best projections from which to approximate (or compress) the cost. After compressing the cost in this way, we can apply the previously-discussed result to restrict our search efforts to a $d+1$ dimensional subspace of the ambient space, in the hope that the true optimal solution will be near this subspace as long as the cost can be compressed adequately in such a way.

2.2.1 Learning low dimensional structure in robot arm planning

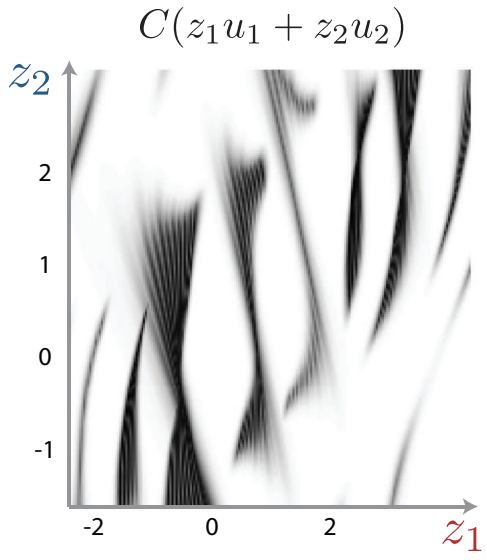
Figure 4 depicts a simple arm planning problem in the presence of obstacles. Our goal will be to find a small set of principal cost components (or *basis motions*) that cause the most variation in the cost function, which in this case is a function that penalizes proximity of the arm to obstacles. We will then consider planning in the subspace spanned by these motions and the linear interpolant between the start and goal locations.

Fig. 4(a) shows the best two basis motions (denoted u_1 and u_2) learned by our method for the particular obstacle field depicted. The first performs a curling action that emphasizes motion of the joints near the base. As desired, this motion seems to cause the greatest variation in the cost function. The second motion again rotates the joints near the base forward, but it simultaneously counter-rotates the joints further along the length of the arm, causing the arm to fold in on itself.

Fig. 4(b) is an illustration of the cost function C restricted to the subspace spanned by u_1 and u_2.



(a) Learned basis



(b) Subspace cost

Figure 4: Visualization of learned basis vectors and cost as a function of basis coordinates for an arm planning problem. Basis vectors (denoted u_1 and u_2) are illustrated in 4(a). Each line shows a configuration $z_i u_i$ of the robot arm, for $i \in \{1, 2\}$ and varying values of $z_i \in \mathbb{R}$, superimposed on an obstacle map from which the cost function is derived. Cost function in (z_1, z_2) coordinates is displayed in 4(b). Dark regions are areas of high cost.

The figure makes it evident that nontrivial obstacle-free paths may be found by navigating just in this subspace; however, the subspace may not contain the start and goal locations. This is easily remedied by adding a third basis motion consisting of the linear interpolant between the start and goal points. We can then plan in the resulting three-dimensional subspace to find a path between the endpoints that deviates from linear interpolation in the directions of the salient basis motions.

By the previous discussion, this subspace is guaranteed to contain the optimal solution if the cost varies only in the directions of the basis motions. Unfortunately, this assumption will typically not hold for realistic problems. However, it can be used as the basis for algorithms for approximate motion planning that perform well in practice, as we will show shortly.

2.3 Learning Dimensional Descent

The final refinement of our method is based on the observation that we can improve the previous optimization technique by performing it in an iterative fashion. This yields a method analogous to block coordinate descent in finite-dimensional optimization, except that we cycle through blocks of *dimensions* of the configuration space, sequentially optimizing the projection of the path onto these subspaces. Hence, we refer to this method as Learning Dimensional Descent (LDD [Vernaza and Lee, 2011b]).

The previous section discussed how given the basis for the principal variation of the cost, the optimal solution lies in the subspace containing the basis and the endpoints, and this is equivalent to considering paths that vary from the linear interpolant by variations in the span of said basis. The latter interpretation suggests the following iterative generalization of the previous method: given an initial path, find the best path whose deviations from the initial path lie in the span of some basis. This is illustrated in Fig. 5. By choosing the initial path to be the linear interpolant and the initial basis to be the basis of principal cost variation, it is evident that the first step of this algorithm is equivalent to the previous method. However, we can subsequently improve upon this solution by examining variations of it that lie in another low dimensional subspace, and so on. In this way, subsequent iterations can take into account secondary variations of the cost in directions orthogonal to the principal directions of cost variation. Proceeding in this way usually yields much improved solutions in practice.

3 Related work

In this section, we give a brief overview of how these ideas are related to previous work. We categorize these methods based on their use of low dimensional structure and decompositions, as well as their applicability to general high dimensional planning problems.

3.1 Generic optimal planning in high dimensions

Some of the simplest methods that are applicable to high dimensional planning problems are those that exploit derivative information. The simplest of these, known as the elastic band method [Quinlan and Khatib, 1993], is applicable to the same holonomic optimal motion planning problem as considered here. That method simulates an elastic band that settles into a locally optimal path as it is pushed along its length by the negative gradient of the cost. CHOMP [Ratliff et al., 2009] is a similar method based on second-order optimization. Another class of local optimization methods are those inspired by Differential Dynamic Programming (DDP [Jacobson and Mayne, 1970]), which at each iteration builds a sequence of approximate quadratic value functions along the length of the current path. These value functions are obtained by recursively applying Bellman's equation along the path, and a new path is then found by simulating the resulting optimal controls. More modern and general variations feature improved convergence rate (as in iLQR [Li and Todorov, 2004]) and applicability to constrained, stochastic systems (as in iLQG [Todorov and Li, 2005]). LDD is significantly different from all of these methods in that it performs *global* dynamic programming at each iteration with respect to what can be considered a simplified cost function, as opposed to performing a local adjustment to the path, or applying dynamic programming only locally. Doing so enables paths in successive iterations to easily transition between homotopy classes, which improves the probability of finding a high-quality local minimum.

An example of a generic method that does not employ gradient information is the recently developed RRT* [Karaman and Frazzoli, 2010, Karaman and Frazzoli, 2011], which is a sampling based method similar to RRT [LaValle and Kuffner, 1999]. Unlike the RRT and most other sampling based methods,

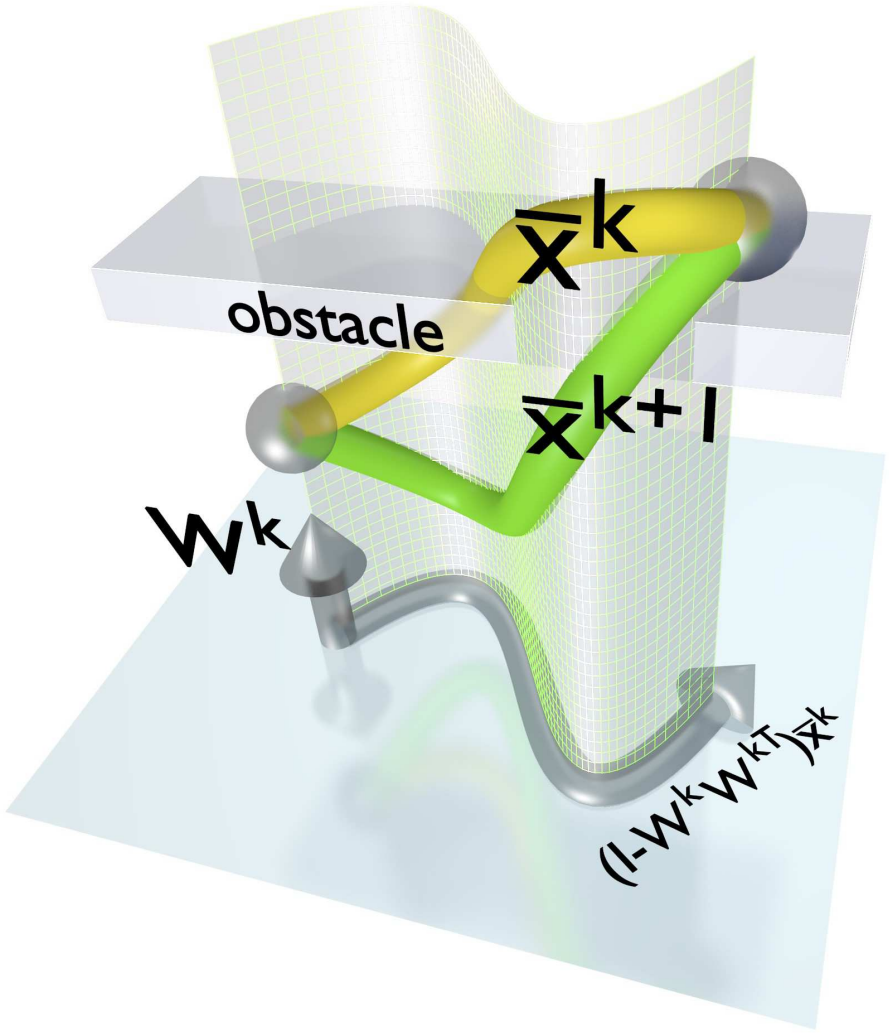


Figure 5: Illustration of one step of dimensional descent algorithm. Given initial path $\bar{x}^k$, the next path $\bar{x}^{i+1}$ is the optimal path constrained to lie on a manifold obtained by sweeping $\bar{x}^k$ along the learned basis direction W^k. This manifold is conveniently parameterized by the curvilinear coordinates induced by the tangent vectors W^k and $(I - W^k W^{kT})\bar{x}^k(\cdot)$. In this illustration, $\bar{x}^k$ is in collision with an obstacle, but optimization over the submanifold yields the collision-free path $\bar{x}^{k+1}$

however, RRT* it is guaranteed to find an optimal solution asymptotically. This is achieved by insisting that new nodes be connected to the tree via a minimum-cost tree path and subsequently reconnecting neighboring nodes as needed, in a manner that is reminiscent of classical graph-based search algorithms.

3.2 Planning with low dimensional structure

Planning aided by low dimensional structure has been a topic of recent interest in the sampling based motion planning literature. Several randomized planning methods have appeared employing low dimensional auxiliary spaces to aid in creating a sort of sampling bias towards the most important degrees of freedom [Shkolnik and Tedrake, 2009, Diankov et al., 2008, Berenson et al., 2009, Sucan and Kavraki, 2008]. Exemplary among these are BiSpace Planning [Diankov et al., 2008] and Task-space RRT [Shkolnik and Tedrake, 2009], which both grow trees in a low dimensional, user-defined workspace or *task space*. This approach has been demonstrated to outperform planning in configuration space by orders of magnitude [Shkolnik and Tedrake, 2009]. Little work so far has been devoted to automatically identifying these interesting low dimensional subspaces. One notable exception is [Dalibard and Laumond, 2009], which uses PCA to determine directions in which to expand an RRT.

The idea of exploiting compression for planning has received more attention in the field of reinforcement learning, where it has been employed in the context of planning for partially observable Markov decision processes (POMDPs). *Belief compression* [Roy and Gordon, 2003] and *value-directed compression* [Poupart and Boutilier, 2003] are among the most well-known of these techniques. Both methods obtain (discrete-state) compressed POMDPs in an automatic way and subsequently plan efficiently in these smaller representations.

3.3 Decompositional and hierarchical methods

LDD may be motivated in various ways, not the least of which is a desire to somehow factorize the planning problem into more easily-solved subproblems. A classical decomposition based approach is described in [Brock and Kavraki, 2001], in which a plan is generated in a high dimensional configuration space given a plan in a low dimensional workspace. Further examples of this arise in legged locomotion, where a number of approaches are based on some variation of the following program: first plan a trajectory for the robot's center of mass, subsequently find a high dimensional plan in the space of footstep locations to follow this path, and finally generate a plan in the high dimensional configuration space of the robot that is compatible with the footstep plan [Kolter et al., 2008, Zucker et al., 2010, Hauser et al., 2008, Rebula et al., 2007].

The success of such methods generally hinges on finding good decompositions that allow the later, higher-dimensional plans to be generated using simpler methods that are likely—but not guaranteed—to succeed if initialized well. Such decompositions are built manually based on the prior knowledge of a designer. From the perspective of machine learning, or the practitioner tasked with building such systems, this degree of manual intervention is very undesirable. An ideal planning algorithm would perform these decompositions automatically based on an examination of the problem data, adapting itself to different scenarios with minimal human intervention.

Some previous work along these lines comes from the literature in deterministic search, where *abstraction hierarchies* have been employed to accelerate search in different ways. Hierarchical A* [Holte et al., 1996] automatically builds hierarchies of abstractions of discrete planning problems. The heuristic of a state at a given level of abstraction is computed as the optimal value of the state in the next-higher level of abstraction. A similar method [Bulitko et al., 2005] also builds hierarchies of abstractions, but employs them differently; instead of using the hierarchy to compute heuristics, the hierarchy is used to plan in a coarse-to-fine manner, first generating a plan at the most abstract level, and subsequently generating more detailed plans at more concrete levels. Although these methods derive from a similar philosophy as that which motivates LDD, they differ fundamentally in that LDD is able to leverage the continuous structure of motion planning problems to find more suitable abstractions.

LDD learns an (ordered) basis that enables and automates the idea of planning first for the most important degrees of freedom and only later planning for the less important degrees of freedom, once the most critical aspects of the path have been determined. The *learning* part of LDD hence consists of finding the basis, while the *dimensional descent* part uses the basis to sequentially plan through the dimensions of the learned basis, in order of descending importance.

It is notable that the method of *Variational Dynamic Programming* (VDP) previously employed an idea very similar to dimensional descent, albeit using a changing, completely random basis in lieu of a learned basis [Barraquand and Ferbach, 1994]. As a result, VDP does not admit the same theoretical optimality guarantees that LDD possesses when applied to problems with low dimensional cost functions. Furthermore, the experiments in Section 6.2 show that the choice of basis is crucial in practice, since a poor basis may result in early convergence to a poor local minimum. Lastly, VDP is not guaranteed to produce monotonic improvement in the continuum limit, in contrast to LDD, because it does not account for important metric transformations between iterations, as described in Section 6.1.1.

4 Exploiting exact low dimensional structure

The notion of low dimensional cost structure in holonomic motion planning can be formalized as follows. Let $x : I \to \mathbb{R}^N$ denote a trajectory of some entity as a differentiable function from time (spanning the unit interval I) to the entity's configuration space, which is assumed to be $\mathbb{R}^N$ for the time being. Our goal will be to find a trajectory $x^*(t)$ such that $x^*(0) = x_a$ and $x^*(1) = x_b$, for some specified $x_a, x_b \in \mathbb{R}^N$, that minimizes a certain cost functional corresponding to the intuitive notion of a *least-cost path* under a given *cost function* $C : \mathbb{R}^N \to \mathbb{R}$. Given these definitions, we formally define the problem of finding an optimal trajectory $x^*(t)$ as follows.

Definition 4.1 (Holonomic optimal motion planning)**.**

$$J\{x\} := \int_0^1 C(x(t))\|\dot{x}(t)\|dt \tag{2}$$

$$x^* := \arg\min_x J\{x\} \tag{3}$$

$$\begin{aligned} \textit{subject to} \quad & x(0) = x_a \\ & x(1) = x_b \end{aligned}$$

The cost functional J integrates cost along a path in a way that is dependent only on the shape of the path and not its specific time parameterization [Do Carmo, 1976]. Intuitively, the $\|\dot{x}\|$ term in the integrand ensures that the path cannot avoid accumulating cost in high cost regions simply by increasing its speed in those regions; it must avoid them instead.

Another salient feature of Definition 4.1 is that it assumes only holonomic constraints are present. We define a *holonomic constraint* to be a constraint of the form $f(x) = 0$, which must be satisfied for all x along the path. Solutions to (4.1) can be made to satisfy such a constraint to arbitrary precision by optimizing a cost function $C(x) + P(f(x))$, where $P(\cdot)$ is a suitable penalty function. Here we consider only holonomic constraints; extensions of our method to deal with nonholonomic constraints will be explored in future work.

The optimization (4.1) may be solved via dynamic programming (DP); specifically, the Fast Marching Method (FMM [Sethian, 1996]), which is very similar to Dijkstra's algorithm in practice, but with a different Bellman update. This update is based on the Hamilton-Jacobi-Bellman equation of continuous optimal control, rather than the Bellman equation of dynamic programming in discrete spaces [Bertsekas, 2005]. The particular equation, known in optics as the Eikonal equation [Hecht, 2001], is a partial differential equation that is approximated via finite differences on a sampled lattice. This discretization induces a quadratic update equation that is used to update the optimal value in the FMM, in contrast to the non-differentiable Bellman equation upon which Dijkstra's algorithm is based.

The computational complexity of DP is nearly linear in the number of points at which we sample our domain. Unfortunately, in order to obtain a good sampling of most domains, we must sample a number of points that grows exponentially with the dimension of the domain, making this a computationally intractable approach for high dimensional problems.

4.1 The virtue of low dimensional cost structure

We now present several equivalent results that reveal in different ways how low-dimensional structure alleviates the curse of dimensionality. The first is a formalization of that which was previously illustrated in Fig. 1. In the following, we assume that C has been augmented by appropriate penalties to enforce holonomic constraints, as described above.

Theorem 4.1. *Consider a holonomic motion planning problem (4.1) over $\mathbb{R}^N$ with a cost function $C : \mathbb{R}^N \to \mathbb{R}$, and suppose that there exists an $N \times d$ matrix W such that $d \leq N$ and*

$$C(y) = C(WW^T y),\ \forall y \in \mathbb{R}^N.$$

Let $I = [0, 1]$ denote the unit interval. Then there exist an optimal path $x^* : I \to \mathbb{R}^N$ of this planning problem and functions $a : I \to \mathbb{R}^d$, $s : I \to \mathbb{R}$ such that

$$x^*(t) = Wa(t) + x_a + (x_b - x_a)s(t),\ \forall t \in I \tag{4}$$

Again, this is the result illustrated in Fig. 1. Intuitively, it means that a d-dimensional cost function (i.e., one such that the cost of any point can be predicted perfectly by its projection onto a d-dimensional subspace) is associated with optimal paths that do not stray outside a known $(d+1)$-dimensional subspace. A proof is given in the appendix. Alternatively, we can loosely characterize the optimal paths found thus as *deviating from the linear interpolant only in the directions upon which the cost depends*, as stated in Eq. (4). This particular characterization is worth remembering, as it lends itself well to further generalization, as will be related shortly.

An equally insightful view may be obtained by considering the optimal value function associated with this problem. Specifically, suppose that we fix $x(0) = 0$ and consider the optimal value $V(y)$ as a function of the other endpoint $x(1) = y$:

$$V(y) = \min_x J\{x\} \text{ s.t. } x(0) = 0,\ x(1) = y. \tag{5}$$

Then the following result holds.

Theorem 4.2. *Suppose that C is the cost associated with a variational problem of the form (2) and V is the associated value function. If W is such that $C(x) = C(WW^T x), \forall x \in \mathbb{R}^N$, and R satisfies $RW = W$, $R^T R = I$, then $V(y) = V(Ry), \forall y \in \mathbb{R}^N$.*

Proof. Let $z(t) = Rx(t)$. The proof follows by algebra:

$$\begin{aligned}
V(y) &= \min_x \int_0^1 \|\dot{x}(t)\| C(x(t))dt, \text{ s.t. } x(0) = 0,\ x(1) = y \\
&= \min_z \int_0^1 \|\dot{z}(t)\| C(R^T z(t))dt, \text{ s.t. } z(0) = 0,\ z(1) = Ry \\
&= \min_z \int_0^1 \|\dot{z}(t)\| C(WW^T R^T z(t))dt, \text{ s.t. } z(0) = 0,\ z(1) = Ry \\
&= \min_z \int_0^1 \|\dot{z}(t)\| C(WW^T z(t))dt, \text{ s.t. } z(0) = 0,\ z(1) = Ry \\
&= \min_z \int_0^1 \|\dot{z}(t)\| C(z(t))dt, \text{ s.t. } z(0) = 0,\ z(1) = Ry \\
&= V(Ry).
\end{aligned} \tag{6}$$

□

Thus, the optimal value function is symmetric about rotations that preserve the subspace in which the cost varies. An important consequence of this is that the value function can be compressed from an N-dimensional object to a $(d + 1)$-dimensional object, where d is the dimension of the cost function.

Corollary 4.3 (Value function compression)**.** *Suppose that C is the cost associated with a variational problem of the form (2) and V is the associated value function. If W is such that $C(y) = C(WW^T y), \forall y \in \mathbb{R}^N$; and ν is any vector such that $W^T \nu = 0$ and $\|\nu\| = 1$, then*

$$V(y) = V(WW^T y + \|(I - WW^T)y\|\nu),\ \forall y \in \mathbb{R}^N. \tag{7}$$

Proof. We proceed by constructing a rotation satisfying the conditions of Theorem 4.2 that rotates y onto the subspace spanned by W and ν. Defining

$$y_\perp = (I - WW^T)y, \tag{8}$$

we choose R to be any rotation satisfying

$$\begin{aligned} RW &= W \\ R\frac{y_\perp}{\|y_\perp\|} &= \nu. \end{aligned} \tag{9}$$

Note that the orthonormality assumptions on ν render this a valid rotation. Then

$$\begin{aligned} Ry &= R(WW^T y + (I - WW^T)y) && (10) \\ &= WW^T y + \|y_\perp\|\nu. && (11) \end{aligned}$$

Applying Theorem 4.2 to this expression yields the desired conclusion. □

The compressed representation of the value function therefore consists of the restriction of the value function to a subspace spanned by W and an arbitrary vector ν orthogonal to it. The value at an arbitrary point y may be extracted by rotating y onto this subspace and evaluating the restricted value function at this new point. Thus, if W is an $N \times d$ matrix, we have reduced the problem of computing a N-dimensional value function to that of computing a $d+1$-dimensional value function.

5 Learning cost structure

Therefore, given a problem with low dimensional structure and knowledge of this structure, we can apply any of the previous results to efficiently solve it. This section describes in detail the approach mentioned in Section 2.2, which is known as Spectral Learning of Approximate Symmetries for high dimensional Dynamic Programming (SLASHDP [Vernaza and Lee, 2011a]).

SLASHDP is best motivated by Corollary 4.3, which states that if a cost function is exactly low dimensional, then the corresponding value function can be represented exactly in a compressed form; expressed differently, exact compressibility of the cost function implies exact compressibility of the value function. If, however, the cost function is not exactly compressible, but we can obtain a good lossy compression of it, then we expect that we can likewise obtain a good lossy compression of the value function by applying Corollary 4.3 anyway. SLASHDP consists of exactly this strategy, along with a method of compressing the cost function by finding the directions in which it varies principally.

5.1 Spectral learning of cost structure

SLASHDP applies to the holonomic optimal motion planning problem previously described. Recalling that discussion, dynamic programming for this problem involves finding a value function $V(y)$ mapping states to the minimum cost of a path starting at y and ending at the goal. The value function is defined as

$$\begin{aligned} V(y) := \min_x J\{x\} \quad &= \quad \int_0^1 \|\dot{x}(t)\| C(x(t)) dt \\ \text{subject to} \quad & x(0) = 0 \\ & x(1) = y, \end{aligned} \tag{12}$$

where $C(x)$ is a cost function assumed to be given as part of the problem specification. Due to the holonomic assumption, an optimal control can be found by integrating

$$\dot{x}(t) = \alpha \nabla V(x(t)), \tag{13}$$

backwards in time from $x(1) = y$, where α is a constant that may be chosen retroactively to ensure $x(0) = 0$ [Bertsekas, 2005].[1]

As previously discussed, the infeasibility of direct computation of the value function motivates a solution based on finding a compressed representation of the cost function, from which a corresponding compressed representation of the value function can be obtained efficiently. This can be achieved in practice by applying Corollary 4.3.

[1] Note that any curve that is a time reparameterization of x will also be optimal since $J\{x\}$ is invariant with respect to time-reparameterization of x.

5.1.1 Estimating the cost basis

For such a scheme to prove useful in practice, we must have that $d \ll N$, since computing the value function will otherwise be infeasible. In the common case that no such W exists, we can still apply this general methodology; however, instead of exactly compressing C via the relation $C(x) = C(WW^T x)$, we will apply a *lossy* compression scheme for which this relation will only be approximately true.

In the following, we will make the simplifying assumption that C is at least once-differentiable. However, we note that this assumption is not intrinsically necessary; in the non-differentiable case, a similar method might be motivated equally well by replacing the gradient of C in the following by a finite-difference approximation.

Let us assume that $W^T W = I$. The differentiability assumption along with $C(x) = C(WW^T x), \forall x \in \mathbb{R}^N$ implies that $\forall x \in \mathbb{R}^N$,

$$\begin{aligned} \nabla C(x) &= WW^T \nabla C(WW^T x) \\ (I - WW^T)\nabla C(x) &= (I - WW^T)WW^T \nabla C(WW^T x) \\ (I - WW^T)\nabla C(x) &= 0, \end{aligned} \tag{14}$$

In the general case that the relation $C(x) = C(WW^T x)$ does not hold exactly, a natural idea is to write

$$\nabla C(x) = WW^T \nabla C(x) + (I - WW^T)\nabla C(x), \tag{15}$$

and choose W such that $\|(I - WW^T)\nabla C(x)\|^2$ is as small as possible in expectation with respect to a given sampling distribution—for instance, uniform over a hyperrectangle representing the set of feasible configurations. Equivalently, we can choose so that $\|WW^T \nabla C(x)\|^2$ is as large as possible, while limiting the number of columns d of W:

$$W := \underset{\substack{W^T W = I \\ \operatorname{rank} W = d}}{\arg\max}\, \mathbb{E}_x \|WW^T \nabla C(x)\|^2. \tag{16}$$

This may be approached using standard optimization techniques. Applying a number of transformations, and defining w_i to be the ith column of W, we obtain

$$\begin{aligned} \mathbb{E}_x \|WW^T \nabla C(x)\|^2 &= \mathbb{E}_x \nabla C(x)^T WW^T \nabla C(x) \\ &= \sum_{i=1}^{d} \mathbb{E}_x \nabla C(x)^T w_i w_i^T \nabla C(x) \qquad (17) \\ &= \sum_{i=1}^{d} w_i^T (\mathbb{E}_x \nabla C(x) \nabla C(x)^T) w_i. \qquad (18) \end{aligned}$$

Adding Lagrange multipliers λ_i to enforce the constraints $\|w_i\|^2 = 1$ and differentiating with respect to each w_i yields

$$\mathbb{E}_x \nabla C(x) \nabla C(x)^T w_i = \lambda_i w_i, \ \forall i \in \{1, \ldots, d\}. \tag{19}$$

Back-substitution of this expression into the objective shows that in order to maximize the objective, we should choose the columns of W to be the eigenvectors associated with the largest d eigenvalues of $E_x \nabla C(x) \nabla C(x)^T$. Algorithm 1 summarizes the basis estimation procedure using Matlab indexing notation.

Algorithm 1 EstimateCostCompressionBasis(d)

```
for i = 1 → numberGradientSamples do
  x ← RandomConfigurationSample()
  G(:, i) ← ∇C(x)
end for
W ← TopNEigenvectors(d, GG^T) {Return top d eigenvectors}
return W
```

We refer to $M := \mathbb{E}_x \nabla C(x) \nabla C(x)^T$ as the *matrix of second moments* of the cost gradient. Of course, since M is not known a priori, it must be estimated by sampling. Although sampling in high dimensions is generally difficult, we observe that this particular sampling problem is at least tractable in the case that $C(x) = C(WW^T x), \forall x$. This is due to the fact that the number of nonzero eigenvalues of M is exactly equal to the rank of W. If the rank of W is small, we can easily estimate the corresponding eigenvectors from a small number of samples. Note that this is true without needing to know W a priori.

Once such a basis is obtained, we can proceed as if the cost were exactly compressible by projection onto this basis by applying the results of Section 4. Further implementation details are provided in Section 5.4.

5.2 A generalized shortcut heuristic

SLASHDP can also be applied locally to segments of a given trajectory in order to replace them with improved segments. This variation can be seen as a generalization of the *straight-line shortcut heuristic* [Geraerts and Overmars, 2004, Geraerts and Overmars, 2007] widely used to post-process feasible motion plans generated by sampling-based methods. The straight-line shortcut heuristic might be described in this way: connect two points on a path with a straight line; if the cost of this path is lower than the original segment, replace it with a straight line. If our cost function is constant in the free space, then a straight line is the optimal path. The straight-line shortcut method can then be seen as a special case that arises when a zero-dimensional (constant) approximation of the cost function is used. SLASHDP, however, can employ higher-dimensional cost approximations, resulting in *higher-dimensional, nonlinear* shortcuts—using a d-dimensional cost approximation, we can search over a space of paths spanning a $d+1$-dimensional space.

Furthermore, SLASHDP shortcuts can be viewed as a more principled version of the *partial shortcut heuristic* proposed in [Geraerts and Overmars, 2004, Geraerts and Overmars, 2007]. That method is based on the observation that it may sometimes be preferable to attempt a linear shortcut in some degrees of freedom while leaving the others untouched, since a shortcut in all degrees of freedom is generally more likely to fail. This behavior emerges in SLASHDP shortcuts as a simple result of finding an optimal path with respect to a locally low dimensional cost function; as previously discussed, such a path will deviate from the linear interpolant only in the directions in which the cost varies principally (Theorem 4.1). Hence, most of the degrees of freedom (in some coordinates) will correspond to linear interpolation, while a few important degrees of freedom will vary in such a way as to minimize cost.

5.3 Results

For the purpose of evaluation, SLASHDP was applied to two problems: planning for a robot arm with many degrees of freedom and planning for a deformable robot, as described in the following section. Though we will see that LDD generalizes SLASHDP for most purposes, we find it insightful to examine how the simpler SLASHDP performs on these problems.

5.3.1 Planning for a planar arm

For this experiment, planar arms with varying numbers of joints were simulated. SLASHDP was applied to optimize a maximum-clearance-type objective that exponentially penalizes proximity to obstacles. Specifically, paths $q(t)$ in joint angle space were sought to optimize the cost functional

$$J[q] = \int_0^1 (1 + e^{-(d_{obs}(q(t)) - d_0)/\bar{d}}) \|\dot{q}\| dt$$

where $d_{obs}(q)$ is the nearest distance to an obstacle when the arm is in configuration q, and d_0 and $\bar{d}$ are fixed parameters.

Fig. 6 shows qualitative results from this problem comparing SLASHDP to a standard bidirectional RRT. Each sub-figure shows the set of points swept out by a 36-dimensional arm as it travels along the solution trajectory obtained with each method. The RRT produces a collision-free path, but it is very complicated. The arm sweeps out a large area as it travels along this trajectory, and it comes very close to collision many times. By contrast, the SLASHDP solution is smooth and collision-free while maintaining a large amount of clearance to all obstacles.

(a) RRT (b) SLASHDP

Figure 6: Subjective comparison of different methods applied to a 36-dimensional arm planning problem. Black shapes represent obstacles. Colored/shaded areas represent set of points visited by each method's solution path. RRT produces a feasible solution, but it is complicated and passes near many obstacles. SLASHDP generates a high-quality solution that is smooth and maintains a large amount of clearance to obstacles.

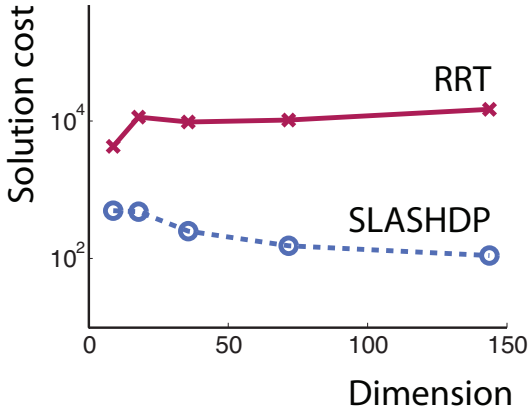


Figure 7: Quantitative comparison of SLASHDP and RRT in planar arm planning experiment showing solution cost scaling as a function of problem dimension.

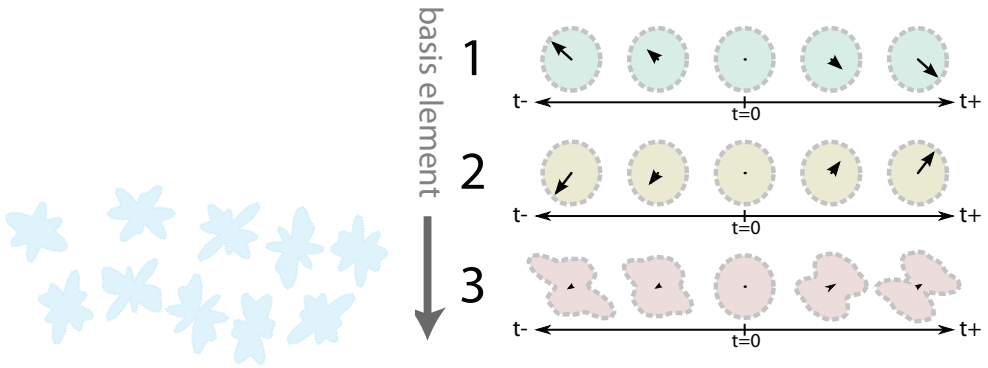
Fig. 7 shows how the quality of solutions obtained via both methods scales as a function of dimension. As one might expect, the quality of the RRT solutions generally degrades slightly as dimensionality increases, since it performs no optimization. The opposite trend, however, is seen in the SLASHDP solutions. This figure also demonstrates that SLASHDP was able to find high-quality solutions for arms with 144 degrees of freedom. This was achieved by performing DP in a four-dimensional subspace.

5.3.2 Planning for a deformable robot

SLASHDP was also evaluated on a high dimensional deformable-robot planning problem. The robot is assumed to live in a two-dimensional space, where it can translate freely and deform in a way that is controlled by a high dimensional set of configuration parameters q. Specifically, the robot boundary is given by a function $r(\theta, q)$ that gives the distance of the boundary from a reference point at angle θ, when the robot is in configuration q. This function is given as a Fourier series expansion:

$$r(\theta, q) = r_0 + \sum_{k=1}^{N} s_{2k} q_{2k} \cos k\theta + s_{2k+1} q_{2k+1} \sin k\theta$$

where the s_k are constant scale parameters.



(a) Random configurations (b) Learned basis

Figure 8: Visualization of random configurations vs. learned basis for deformable robot planning problem. Randomly sampled configurations are shown for reference in 8(a). Learned basis vectors are displayed in 8(b). Each row represents one of three learned basis vectors. Shape in each column represents the shape with coordinates tu_i, where u_i is the ith learned basis vector. Central arrows show magnitude and magnitude of translation. First two basis vectors encode position of robot with no deformation. Third basis vector encodes a useful deformation of robot with no translation.

The position $x(\theta, q)$ of a point on the robot boundary is then given as

$$x(\theta, q) = \begin{pmatrix} q_{2N+2} \\ q_{2N+3} \end{pmatrix} + r(\theta, q) \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}.$$

A cost function $C(x)$ is defined to penalize the proximity of any point on the boundary to an obstacle:

$$C(x) = 1 + e^{-(d_{obs}(x) - d_0)/\bar{d}}.$$

For computational purposes, K evenly-spaced $\theta_k \in [0, 2\pi)$ were chosen along the boundary of the robot. Let $x_k(q) = r(\theta_k, q)$ denote the position of the kth sample. Given this notation, we can write the cost functional as

$$J[q] = \int_0^1 \left(\sum_{k=1}^{K} C(x_k(q)) \right) \|\dot{q}\| dt.$$

SLASHDP was then applied to the problem in this form.

Fig. 8 shows the three-dimensional basis learned for a specific instance of this problem consisting of a maze-like environment. As would be expected, the first two basis vectors encode just the position of the robot with no deformation, while the last encodes a useful-looking deformation with no translation component. To understand why this is so, it is useful to view the nature of the maze depicted in Fig. 9, which consists of corridors running in the "northeast-southwest" and "northwest-southeast" directions. It is clear that the third basis vector encodes a deformation that the robot to travel through either type of corridor with low cost.

Qualitative results are depicted in Fig. 9. It is readily apparent how the solution plans are constructed as deviations from the linear interpolant along the three basis directions. It is also apparent that in order to achieve better performance, the plan must be allowed to vary along more dimensions than just the three learned basis directions; however, simultaneously planning for more than a few directions would be prohibitively expensive. In Section 6, we will see how transforming SLASHDP into an iterative algorithm allows us to achieve superior performance on this problem without excessive computational expense.

5.4 Implementation details

This section reviews some of the more salient implementation details omitted so far; specifically, we describe how to proceed to apply the results of Section 4 once we have obtained a basis for compression of the cost as described in Section 5.1. We then discuss the method's computational performance characteristics.

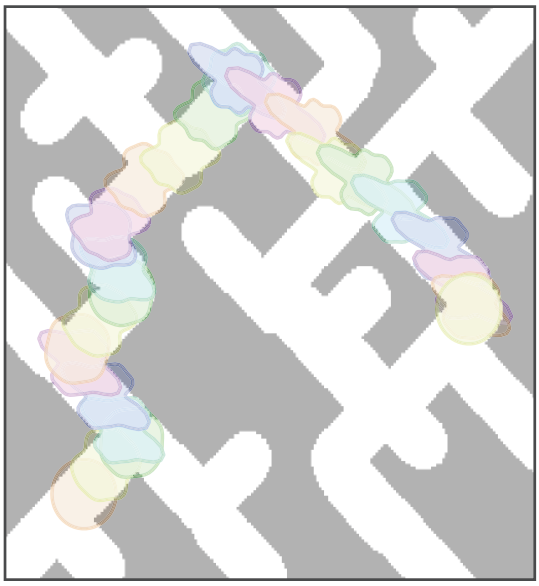


Figure 9: Qualitative results of deformable robot planning with SLASHDP

5.4.1 Compressing the cost function

Once we have learned a basis W via the method described in Sec. 5.1.1, we must estimate the compressed cost function in order to evaluate the compressed value function. That is, writing $z = W^T x$, we wish to find $C_z(z) \approx C(Wz)$, where $C_z : \mathbb{R}^d \to \mathbb{R}$ is the compressed cost function. We then proceed by discretizing z coordinates on a regular grid, which will eventually enable us to compute the value function using a fast, grid-based version of the FMM. For each of these points $z[i]$, we must obtain an estimate of $C_z(z[i])$. This is very simple in the ideal case; we simply take $C_z(z) = C(Wz[i])$.

In the likely event that $\exists x \mid C(x) \neq C(WW^T x)$, different states that project to the same z coordinates can have different cost values—although we intentionally chose our basis to minimize the expected difference, in some sense. The obvious solution to this problem is to sample the costs of states projecting to the same coordinates z, and to set $C_z(z)$ to the mean, max, or min of all such values. Alternatively, supposing for now that we are only interested in finding the optimal path to the goal for a particular *query* state, we know by Theorem 4.1 that the optimal path with respect to the compressed cost function will be contained in the $(d+1)$-dimensional subspace spanned by the columns of W and the query state. Therefore, we can just as well restrict our sampling efforts to this subspace, thus obtaining a far simpler sampling problem. In the event that we subsequently use the value function thus obtained to find paths for query states not in this subspace, this strategy may not be optimal, as the resulting paths will not lie in this subspace; however, this was not found to be a significant issue experimentally, provided that the query configurations were not too far from that assumed in the computation of the compressed cost.

5.4.2 Computational considerations

We note that there are three major components to the computational cost of SLASHDP: sampling cost gradients, sampling the cost function in the reduced space, and performing dynamic programming. For a d-dimensional compressed cost using k samples per dimension, sampling the cost function takes time $O(k^d)$, and dynamic programming takes time $O(k^{d+1}(d+1)\log k)$. If we make the reasonable assumption that we will not need to sample more gradients than there are samples in our grid, then sampling the gradients also takes time $O(k^d)$. In practice, any one of these factors may dominate the computation time, depending on the relative expense of computing the cost function and cost gradients vs. dynamic programming. However, we note that the cost gradient samples samples may be computed independently and in parallel; therefore, given sufficient processors, we expect the bottleneck to be in the low dimensional dynamic programming step. As it is possible to implement DP very efficiently,

SLASHDP would probably benefit dramatically from extreme parallelization of the sampling component in multi-core or GPU processors.

6 Learning dimensional descent

SLASHDP is based on the intuition of computing a compressed value function from a compressed cost function. This approach is particularly useful as a *multiple-query* or *feedback* method, in the sense that a single lengthy planning phase generates a value function from which a control law can be extracted very efficiently. For the *single-query* variant of the motion planning problem, on the other hand, we can develop a more path-centric approach to obtain much improved results. This method generates a succession of monotonically improving solutions by optimization over a series of low dimensional submanifolds of the ambient space.

From an optimization perspective, the procedure can be thought of as Hilbert-space block coordinate descent in *learned* coordinates. In this setting, we regard the optimization problem as that of finding a *function* (i.e., a path) that optimizes a given cost functional. The space of all functions considered constitutes an infinite-dimensional vector space (or Hilbert space) of functions mapping time to configuration-space coordinates. Block coordinate descent is a basic optimization algorithm, usually applied to finite-dimensional optimization, which in each iteration considers varying the projection of the current vector onto some subspace, while keeping fixed the projection of the vector onto the rest of the space. We proceed similarly—albeit in the infinite dimensional space of paths—by optimizing in each iteration the projection of the path onto some subspace of the configuration space, while keeping fixed the projection of the path onto the rest of the space. To highlight the differences between finite-dimensional block coordinate descent and our method, we refer to our approach as Learning Dimensional Descent (LDD [Vernaza and Lee, 2011b]).

6.1 Method

Here we develop LDD so as to essentially generalize SLASHDP. SLASHDP can be viewed as optimizing the following objective, where $J\{x\}$ is the holonomic-optimal-control cost functional in Eq. (12),

$$\begin{aligned}(a^*, s^*) &:= \arg\min_{a,s} J\{Wa + ys\} \\ x^* &:= Wa^* + ys^*,\end{aligned} \tag{20}$$

and, as before, y is assumed to be the goal, the other endpoint is assumed to be the origin, and W is the basis learned by SLASHDP, as described in Section 5.1.1. We also use the notation Wa to denote the function $t \mapsto Wa(t)$.

Generalizing this to implement the previously described program of dimensional descent is conceptually very simple. Given the learned basis W, instead of searching once over the set of paths described by Eq. 4, we find a sequence of paths $\bar{x}^k$ defined by

$$\begin{aligned}(a^{k+1}, s^{k+1}) &:= \arg\min_{a,s} J\{W^k a + \bar{x}^k \circ s\} \\ \bar{x}^{k+1} &:= W^k a^{k+1} + \bar{x}^k \circ s^{k+1},\end{aligned} \tag{21}$$

where each W^k is a matrix comprised of a strict subset of the columns of W. Assuming that W^k is an $N \times d$ matrix, each step involves solving a $d+1$-dimensional DP problem. The algorithm begins by setting W^0 to the first d columns of W, and by setting $\bar{x}^0(s) = ys$—in that sense, generalizing SLASHDP. Subsequent steps set W^k equal to the next d columns of W, and so on. After all the columns of W are exhausted in this way (or we reach a predefined limit), we again set W^k equal to the first d columns of W. LDD proceeds in this way until convergence or some other termination condition is met.

Choosing the W^k in this way ensures that we choose directions to optimize in order of their relative importance. This intuitively minimizes the chance that LDD will get stuck in some poor local minimum early on. Furthermore, by Theorem 4.1, it ensures that LDD will terminate with a globally optimal solution (up to discretization) in one step provided that the conditions of that theorem are met and we choose a large enough d. Note also that whether or not a global optimum is achieved, LDD has the

important property of monotonic convergence towards a local optimum. This is a simple consequence of the fact that the solution in one iteration is contained within the feasible set of the next iteration.

The geometry of the LDD optimization is illustrated in Fig. 5. The set of paths considered by LDD at each iteration is the (generally non-linear) submanifold of $\mathbb{R}^N$ obtained by sweeping $\bar{x}^k$ in the directions spanned by the column space of W^k. When W^k is a single column, this submanifold is a simple *ruled surface*, as illustrated.

A simple variant of LDD is described in pseudocode in Algorithm 2. The pseudocode neglects issues of discretization and interpolation that are considered uninsightful. Some implementation details, however, are elucidated in the next section.

Algorithm 2 LDD(C, x_0, x_1, K, B)

$\{C$: a cost function $\mathbb{R}^N \to \mathbb{R}\}$
$\{x_s$: start configuration $\in \mathbb{R}^N\}$
$\{x_g$: goal configuration $\in \mathbb{R}^N\}$
$\{K$: number of iterations to run$\}$
$\{B$: optimization block size$\}$
$W \leftarrow$ EstimateCostCompressionBasis(N)
$\bar{x}^0 \leftarrow$ (function $t \mapsto (1-t)x_s + tx_g$)
for $k = 0 \to K-1$ **do**
  {Choose next B columns of W}
  $W^k \leftarrow W(:, \text{WrappedIndexRange}(kB \mod N, ((k+1)B-1) \mod N))$
  {Project path onto orthogonal complement of basis, reparameterize by arc length}
  $\bar{x}^k \leftarrow \text{ArcLengthParameterize}(\bar{x}^k - W^k W^{kT} \bar{x}^k)$
  {Define a cost function in submanifold coordinates}
  $C^k \leftarrow$ (function $(a, s) \mapsto C(W^k a + \bar{x}^k(s))$)
  {Find an optimal path between the endpoints directly in submanifold coordinates, using the FMM}
  $(a^{k+1}, s^{k+1}) \leftarrow \text{FMMFindOptimalPath}(C^k, (W^{kT} x_s, 0), (W^{kT} x_g, \text{ArcLength}(\bar{x}^k)))$
  {Rewrite path in configuration space coordinates}
  $\bar{x}^{k+1} \leftarrow$ (function $t \mapsto W^k a^{k+1}(t) + \bar{x}^k(s^{k+1}(t))$)
end for
return $\bar{x}^{K-1}$

6.1.1 Technical details

The rest of this section is devoted to the subtle technical issue of solving the LDD optimization (21) via dynamic programming. A naive attempt to solve (21) might involve sampling a^k and s on a finite lattice, evaluating the cost function at these points as $C(a^k, s) = C(W^k a^k + \bar{x}^k(s))$, and applying the Euclidean FMM to the resulting problem. However, this will not yield the correct result, due to the aforementioned non-Euclidean geometry of the problem (Fig. 5). If we wish to optimize paths under the original objective, which specifies the Euclidean metric in $\mathbb{R}^N$, we must therefore take into account how this metric transforms under a change to the coordinates (a^k, s) of the submanifold that is the set of feasible paths.

To derive the correct procedure, we must therefore substitute the expression for the path in manifold coordinates $x(t) = W^k a^k(t) + \bar{x}^k(s(t))$ into the Euclidean metric to derive its expression in terms of manifold coordinates. Doing so yields

$$\begin{aligned} \dot{x}^T \dot{x} &= \dot{a}^k(t)^T \dot{a}^k(t) + \dot{a}^k(t)^T (W^k)^T \frac{d}{dt}[\bar{x}^k(s(t))] \\ &\quad + \frac{d}{dt}[\bar{x}^k(s(t))]^T \frac{d}{dt}[\bar{x}^k(s(t))]. \end{aligned} \tag{22}$$

To render optimization with respect to this metric a problem amenable to optimization by standard methods, which generally assume the Euclidean metric, note that we can make some helpful simplifying

assumptions. First, note that we can equivalently parameterize the manifold by replacing $\bar{x}^k$ with $\bar{x}^k - W^k(W^k)^T\bar{x}^k$, yielding

$$x(t) = W^k a^k(t) + (I - W^k(W^k)^T)\bar{x}^k(s(t)). \tag{23}$$

(Note that this choice of manifold coordinates is also illustrated in Fig. 5.) Substitution of this expression into the metric causes the cross-term to cancel, resulting in the following diagonal metric, after simplification. Let $P = I - W^k(W^k)^T$. Then

$$\dot{x}^T\dot{x} \quad = \quad \dot{a}^k(t)^T\dot{a}^k(t) + \dot{s}^2\frac{d\bar{x}^k}{ds}^T P^T P\frac{d\bar{x}^k}{ds}. \tag{24}$$

At this point, we wish to make the coefficient of $\dot{s}^2$ equal to one. Note that we can achieve this simply by assuming that $\bar{x}^k(s)$ has the arc-length parameterization after projection onto the subspace orthogonal to W^k. We must therefore take care to make this assumption in all expressions involving $\bar{x}_i$—specifically when applying (23) and when calculating the cost function in terms of manifold coordinates, a^k and s. Doing so ensures that we can safely apply an efficient Euclidean FMM solver to optimize the correct objective.

6.2 Simulation results

We evaluated LDD in simulation on three high dimensional motion planning problems—planning for a deformable robot, planning for a planar arm in the presence of obstacles, and planning for a three-dimensional arm in a high-fidelity simulation. For all of the results described below, the d parameter was set to one.

6.2.1 Deformable robot planning

We applied LDD to the same deformable robot planning problem to which SLASHDP was applied in Sec. 5.3.2. The deformation was controlled by a set of 16 parameters corresponding to Fourier series coefficients of the radius of the boundary of the robot, as a function of angle. To make the problem more challenging, a random rotation was applied to these coordinates in the following way. Instead of inputting the problem data to LDD in the original q coordinates, the problem data were transformed to coordinates $q' = Rq$ by finding an R such that

$$\begin{aligned}(U + U^T)R &= \Lambda R \\ R^T R &= I\end{aligned} \tag{25}$$

where U was chosen to be a square matrix of independent samples from the uniform distribution on the unit interval, and Λ is a diagonal eigenvalue matrix.

Results are shown in Fig. 10. Dimensional descent in the original, randomized coordinates, was compared to LDD in the learned coordinates; that is, LDD was applied to the problem in random coordinates, which then transformed the problem into learned coordinates. Dimensional descent in the random coordinates produced a very poor solution in which the robot did not correctly deform or translate to fit the maze, whereas LDD correctly translated the robot through the maze, while deforming the shape of the robot to fit the corridors of the maze.

Convergence properties of LDD versus dimensional descent (DD) in the random coordinates are shown as insets in Fig. 10. DD in random coordinates yielded a significant improvement in cost in the first iteration, but no noticeable improvement in subsequent iterations, indicating that DD quickly fell into a poor local minimum. LDD also made the bulk of its progress in its first iteration, but its solution after the first iteration was several orders of magnitude less costly than that of DD. After the first iteration, it continued to make steady improvement, especially up until the 10th iteration. After this iteration, LDD seemed to slowly approach a locally optimal solution.

6.2.2 Planar arm planning

LDD was additionally applied to the problem of planning a collision-free trajectory for an 11-DOF planar arm with no self-collisions. The intent was to thoroughly study the ability of LDD to find high-quality solutions in very cluttered environments, especially in relation to other commonly used approaches.

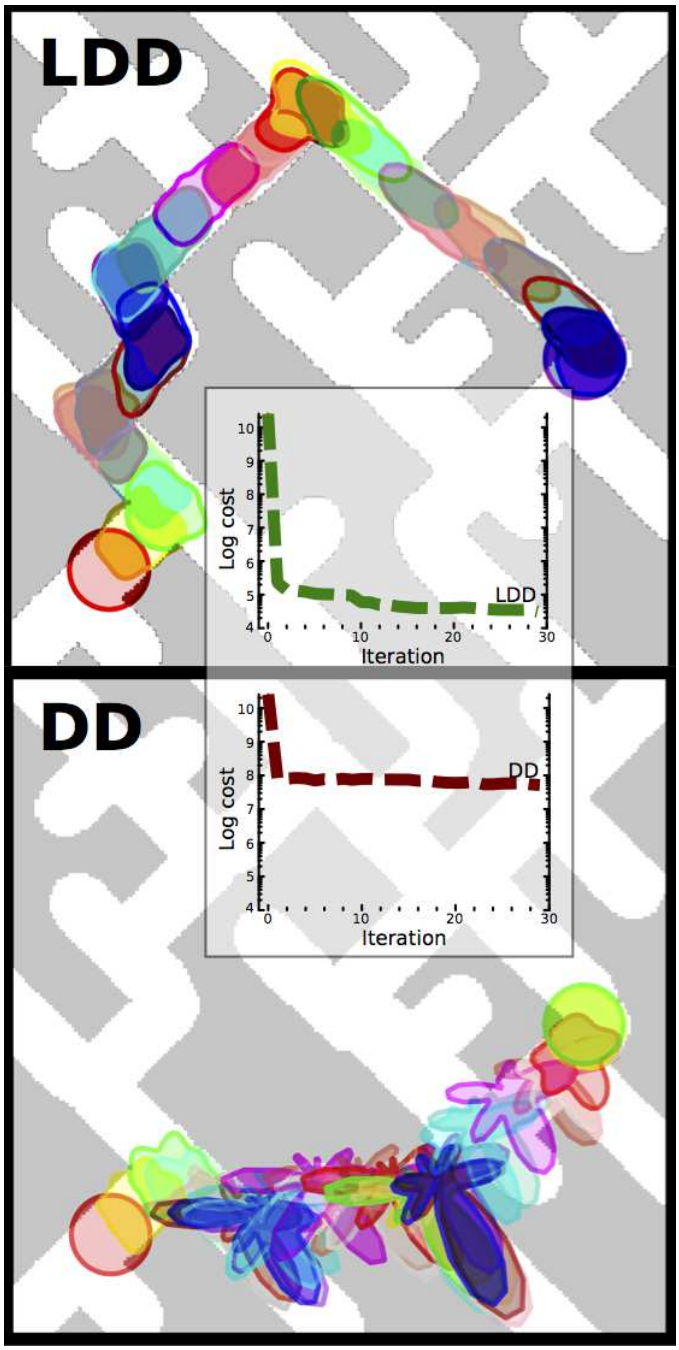


Figure 10: Results of experiment in 18-dimensional shape planning (see text for details). Consecutive shapes shown as overlapping, translucent objects. Insets show log cost as a function of iteration number. Although both methods converge to some solution, LDD converges to a solution several orders of magnitude better than DD.

Figures 11 and 12(a) depict solutions obtained with LDD in highly constrained environments. Fig. 12(a) depicts the scenario used for quantitative evaluation. In this evaluation, collision-free paths were found between given initial and final configurations, while varying the number of obstacles in the environment. The label on each obstacle indicates the trial in which that obstacle was first introduced; each obstacle was then kept in the environment for subsequent trials, making later trials more challenging than earlier ones.

Quantitative results are given in Fig. 12(b). LDD was compared to several other methods, including inflated-heuristic A* with different heuristics and RRT-based planners. Fig. 12(b) shows the cost of the solution obtained with each method for each trial. RRT is a standard bidirectional RRT, provided as a baseline for comparison, though it does not perform any optimization. All other methods attempt some optimization. S-RRT is a bidirectional RRT that has been post-processed with an elastic band planner to improve the quality of its solution, as is common practice. S-TSRRT is a bidirectional variant of Task-Space RRT [Shkolnik and Tedrake, 2009], also post-processed with an elastic band. DD refers to dimensional descent in the original coordinates. A*Proj refers to A* using the heuristic of the distance of the end-effector to the goal, while A*Full is A* using the Euclidean distance heuristic in the configuration space.

Both A* variants and TS-RRT were only able to find solutions for the trials with less than seven obstacles in a reasonable amount of time—these methods ran for more than eight hours on a 3 GHz Intel Xeon test machine without finding a solution. By its nature, DD always found some solution, but it was not always feasible, as evidenced by the fact that the cost of its solutions far exceeded that of the baseline RRT in many trials. The standard bidirectional RRT (pre- and post-smoothing) and LDD were therefore the only methods consistently able to find solutions for the most difficult problems.

In terms of solution quality, LDD outperformed every other method in every trial, usually by a large margin. The performance gap between LDD and S-RRT was as great as a factor of five. For the most difficult trials, LDD still managed to find a solution of roughly half the cost of S-RRT. In order to find any solutions with A* in a reasonable amount of time, a high heuristic weighting factor had to be applied, which generally caused the solutions to be very suboptimal.

6.2.3 High-fidelity arm simulation

LDD was also evaluated in the context of arm planning for the Willow Garage PR2 robot. The PR2 is designed primarily for mobile manipulation tasks, and is equipped with a variety of sensors and two arms, each of which has seven DOF in addition to a one-DOF gripper. This section describes results obtained using a high-fidelity simulation of the PR2.

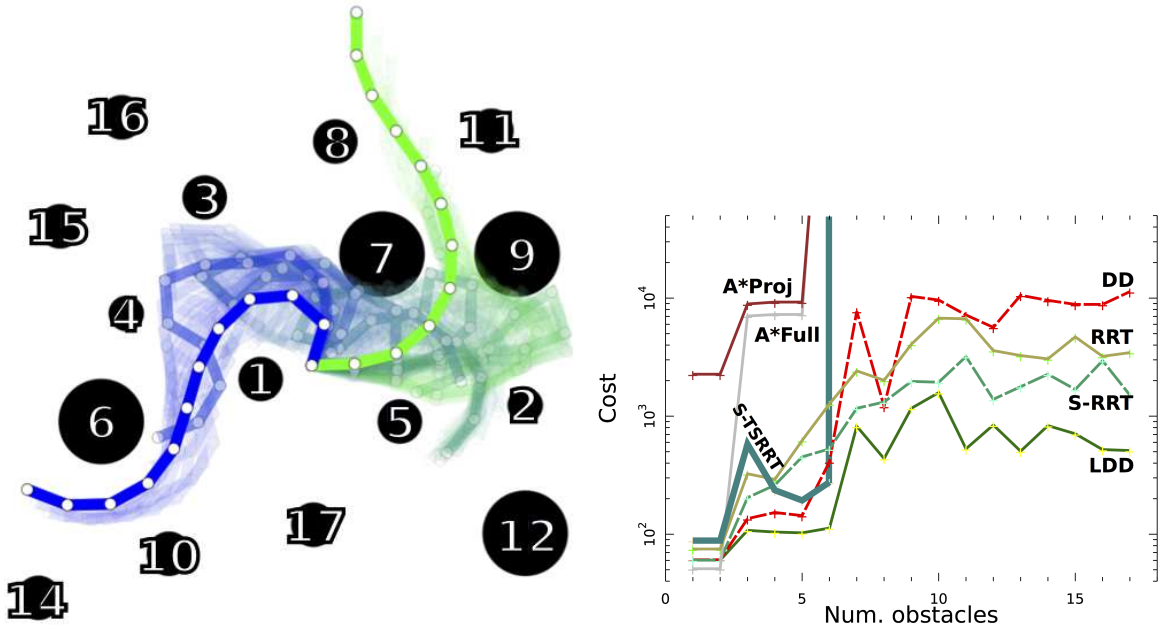
The specific cost function used by LDD was of the form

$$C(q) = 1 + \sum_{i=1}^{n_s} \exp\left(\frac{d_0^i - \min_{o\in\mathcal{O}} \|x^i(q) - o\|}{\bar{d}^i}\right), \tag{26}$$

where $\mathcal{O}$ is the set of points in the robot's workspace that are located within obstacles. n_s samples were chosen uniformly spaced roughly along the medial axis of the robot's arm, with $x^i(q)$ denoting the Cartesian coordinates of the ith sample with the joints in configuration q. Each sample was assigned a soft minimum distance d_0^i and an exponential scale $\bar{d}^i$ based on the approximate radius of the arm at that point. Joint limit constraints were eliminated by clipping invalid joint angles inside their respective bounds (which roughly corresponds to changing to coordinates in which the constraints are always satisfied).

The experiment consisted of attempting to plan a collision-free path for the PR2's right arm between the configurations pictured in Fig. 13. The simulation environment consists of a table upon which a few hollow mailboxes rest. The initial configuration is underneath the table, and the final configuration is inside one of the mailboxes. We compared the planning performance of LDD to that of several other methods: (LB)KPIECE [Sucan and Kavraki, 2008], a state-of-the-art sampling based planner utilizing bidirectional trees and lazy collision checking; and the previously-described CHOMP [Ratliff et al., 2009], and RRT* [Karaman and Frazzoli, 2010]. We additionally investigated the effects of post-processing these methods with a standard trajectory filtering package that employs a combination of cubic splines and linear shortcuts. The experiments employed the open-source OMPL [omp, 2010] implementations of

Figure 11: Sample solution trajectory in highly constrained environment



(a) (b)

Figure 12: Fig. 12(a): visualization of results applying LDD to an arm planning problem. End configurations represented as green and blue solid lines, intermediate configurations by lighter lines. Fig. 12(b): comparison of several methods applied to the problem in 12(a). Abscissa shows number of obstacles, and ordinate shows cost of found solutions (note log scale). Obstacles were added in order shown by 12(a) (obstacle 13 outside area shown). See Sec. 6.2.2 for details.

KPIECE and RRT* [2], and implementations of CHOMP [3] and shortcut/spline-based trajectory filtering [4] distributed as part of the Willow Garage Robot Operating System (ROS[5]).

The objective was mainly to compare the consistency and quality of the solutions obtained with the different methods. Measured arc length of the seven-dimensional joint trajectories was used as a primary metric in order to compare the quality of the solutions. RRT* attempts to optimize the arc-length objective exactly, and the shortcut-based smoother optimizes it only approximately, owing to the fact that it smooths its output with cubic splines. [6] LDD was made to approximately optimize arc length by tuning the cost parameters to obtain as close an approximation to arc length minimization as possible, although the relatively crude collision modeling employed and numerical issues that arise in this limit were limiting factors in this respect.

All methods were run on a computer equipped with a quadruple-core, 2.0 GHz Intel i7 CPU and 8 GB of random access memory. Variations of each method were tried in order to explore different performance characteristics. In order to fairly compare the effectiveness of each optimization method, each method was tried in a variation that forced it to terminate with the best solution found in less than a minute. LDD and LBKPIECE were also tried in variations that simply returned the first feasible solution found. Indicated variations also included post-processing with the aforementioned linear shortcut/cubic spline smoother.

Results are shown in Fig. 14. Ten planning trials with identical start and end configurations were attempted for each method. CHOMP was also the only method tested that was unable to find a collision-free path in any of the 10 trials, as it became stuck in an infeasible local minimum. It is therefore excluded from Fig. 14. RRT* successfully completed 4 of 10 trials in the one-minute time limit set for all methods. All other methods completed all 10 trials successfully.

Predictably, the solutions found by LBKPIECE without post-processing were on average longer than those obtained by any other method. It was the fastest method tested, however, in the sense that a feasible solution was found on average within one second of the start of the trial. Aside from CHOMP, which never found a feasible solution, RRT* was the slowest method, in the sense that in 6 of 10 trials, it was unable to find a solution within a minute. The mean arc length of the RRT* solutions, after a minute of optimization, was 9.8 radians, while the mean arc length of the first feasible solution found by LBKPIECE was 10.0 radians. The RRT* solutions exhibited higher variance than the LBKPIECE solutions, with one apparent outlier among the small number of successful trials.

The LBKPIECE initial feasible solutions followed by one minute of smoothing were on average shorter than those found by one minute of RRT* optimization followed by two seconds of smoothing. The mean solution lengths of the former approach were slightly longer than those found via one minute of LDD optimization. The LDD solutions, however, were much more consistent across trials. Furthermore, smoothing the LDD solutions for an additional two seconds yielded a substantial improvement over smoothing the LBKPIECE solutions for 60 seconds. We believe that this improvement came partially as a result of the aforementioned fact that LDD was directed to only approximately minimize arc length.

In all trials, LDD was set to search over two-dimensional submanifolds in each iteration. For the one-minute trials, LDD was directed to take 100,000 cost gradient samples (in parallel) in order to learn the basis in which to perform the optimization. This resulted in a high degree of consistency across trials, since this estimation procedure is the only nondeterministic component of LDD. Since LDD's speed is highly dependent on this parameter and grid refinement, we additionally tried a variation in which LDD took only 10,000 cost gradient and in which the grid resolution was halved. Plots of the first feasible solutions found thus are shown in the *LDD first feasible* scenario in Fig. 14. The paths produced by this variant, which terminated on average in 3.1 seconds, were only slightly longer on average than those produced by LBKPIECE followed by one minute of smoothing, while exhibiting greater consistency than those of smoothed LBKPIECE.

It is important to note that the timings presented here should only be considered roughly indicative of the potential performance levels of the different methods tested, as the implementations varied significantly in implementation details and optimization techniques. A particular optimization employed

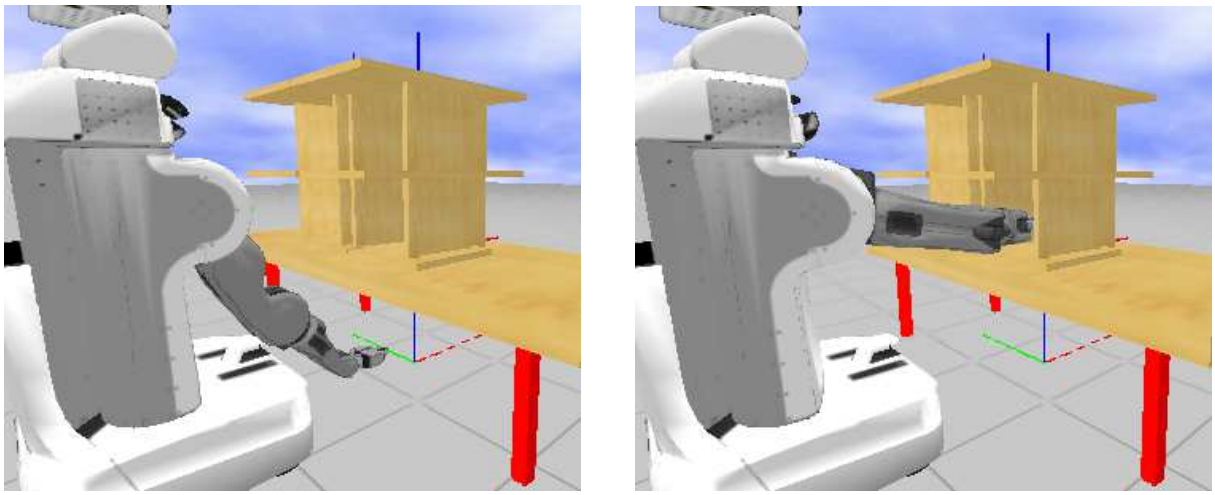
[2]RRT* implementation by Alejandro Perez, Sertac Karaman, and Ioan Sucan

[3]CHOMP implementation by Mrinal Kalakrishnan, available at http://www.ros.org/wiki/chomp_motion_planner

[4]Spline/shortcut trajectory filter available at http://www.ros.org/wiki/constraint_aware_spline_smoother

[5]Available at http://www.ros.org

[6]It is noted that arc length minimization subject to collision constraints is technically ill-posed, as the set of collision-free configurations is an open set; the infimum of the optimization can never be achieved.



(a) Initial (b) Final

Figure 13: Initial and final conditions for high-fidelity simulation experiment.

by LDD and CHOMP, but none of the other methods, was the use of approximate collision checking for the cost computation. In the LDD implementation, feasibility using exact mesh collision checking was only tested after each DP iteration. It is possible that some such optimization would benefit RRT* as well, though it is worth noting that collision checking is not a serious impediment to the performance of LBKPIECE, which shares its collision checking routines with RRT*, and was the fastest of all the methods tested in finding a feasible solution.

6.3 Experimental results

We further evaluated LDD using an actual PR2 robot. Two scenarios in cluttered environments were tested: one in which the gripper was placed in a small window and made to plan a path to another window, and another in which the arm was made to move between two highly-constrained configurations in the midst of pole-like obstacles. LDD was compared to SBL [Sánchez and Latombe, 2001] using the OMPL [omp, 2010] implementation of the latter.

Details of the *windows* experimental setup are shown in Fig. 15. The gripper is roughly 11 cm in width, while both the start and goal configurations were inside windows with a smallest measured side length of 19 cm. For all experiments, the robot's own laser scanner was used to provide an obstacle voxel grid with a resolution of 2 cm.

Some qualitative results of the *windows* experiment are shown in Fig. 16. Although differences between the methods were sometimes difficult to see in real-time, analysis of videos taken during the experiment revealed several cases where it was readily apparent how to improve the smoothed SBL trajectory. In one case, the arm was observed to first back away to the left before proceeding to the goal on the right. In another case, the wrist needlessly rotated the gripper parallel to the ground, despite the fact that the gripper was constrained to be perpendicular to the ground in the start and goal states. By contrast, trajectories generated by LDD could not be improved in an obvious way. Furthermore, the LDD trajectories were basically indistinguishable between trials, giving confidence that the W matrix had been learned well, with little variance.

Qualitative results for the *poles* experiment are depicted in Fig. 17. The observed differences here were often more dramatic and easily visible in real-time. In particular, the smoothed SBL trajectory occasionally took the arm on a circuitous route around and over the pole on the left. The LDD trajectories by comparison were again always very direct and again basically indistinguishable across trials.

Quantitative analysis of the experimental data corroborated these observations, as shown in Fig. 18 (see caption for detailed interpretation). Visualizing the joint trajectories bore out the observation that the LDD trajectories were usually nearly identical across trials. Smoothing the LDD trajectories also

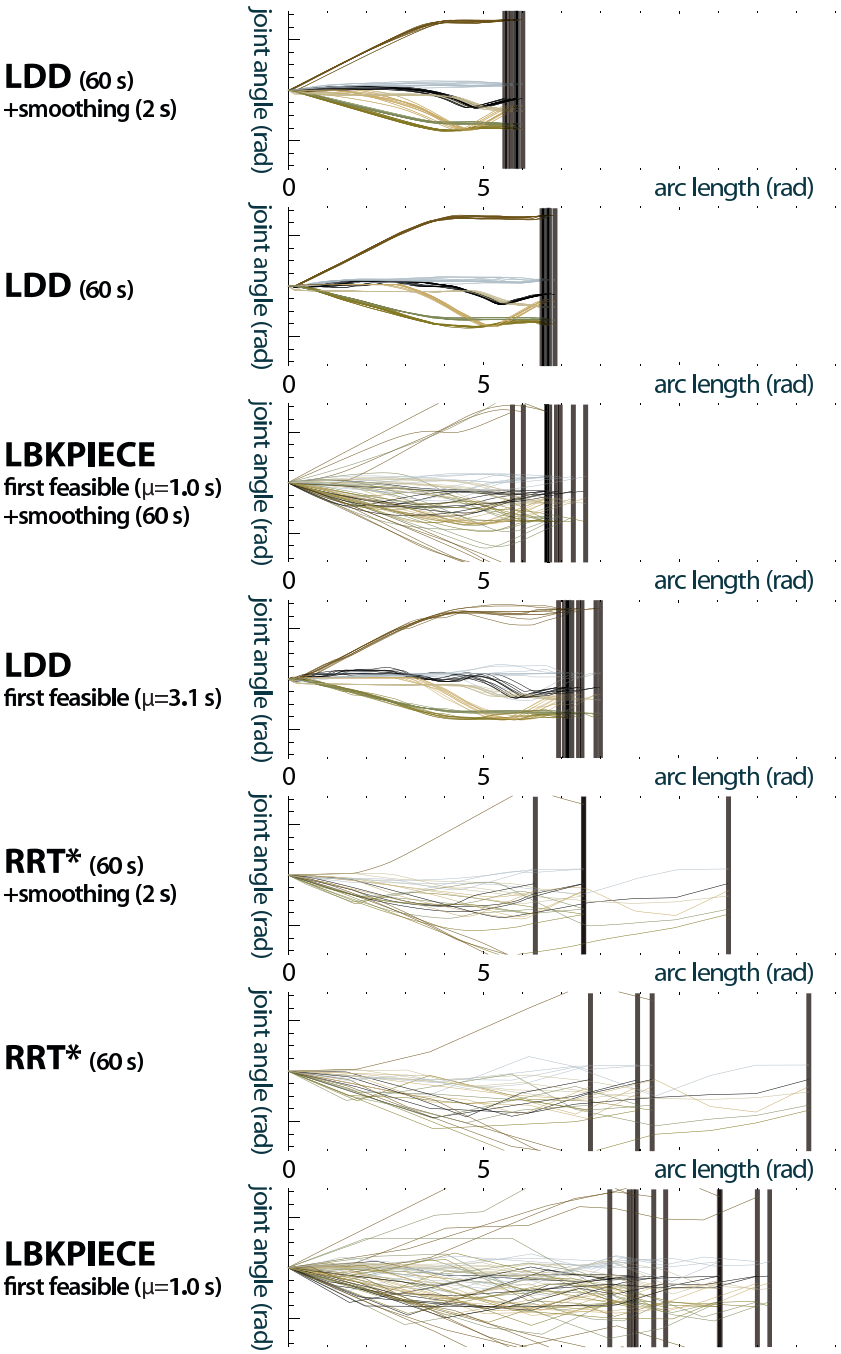


Figure 14: Plots of arc length vs. joint angle obtained in repeated motion planning trials, using several different methods, sorted vertically by mean arc length across trials. Each plot shows the paths of the robot's seven joints as they follow the trajectories generated by the planner with the parameters indicated on the left. The plot of each trial is superimposed on the last, with paths of identical joints colored identically across trials. The total arc length of each trial's path is marked with a thick, vertical line. Each method description indicates the planning method used, the planning time limit set (if any), whether post-smoothing was applied, and the time limit set for post-smoothing, if it was applied. Some timings show the mean time t to the first feasible solution; these are indicated by 'first feasible ($\mu = t$ s)'.

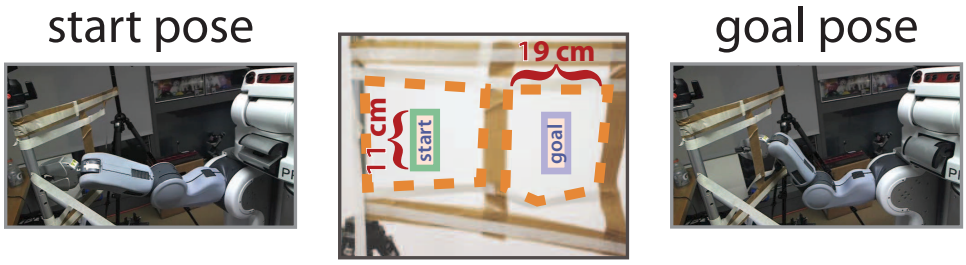
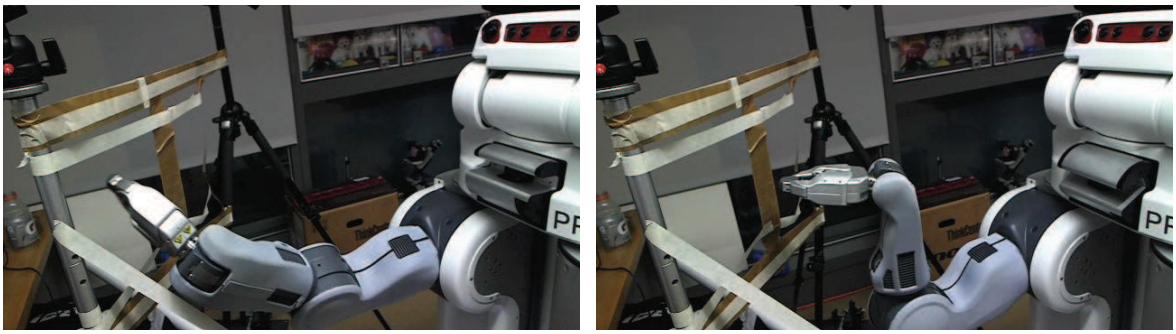


Figure 15: Setup of *windows* experiment. Scale of windows and approximate cross-sections of end-effector at start and goal locations are shown.



(a) (b)

Figure 16: Video stills showing extraneous motions remaining in sampling-based planner's trajectory post-smoothing.

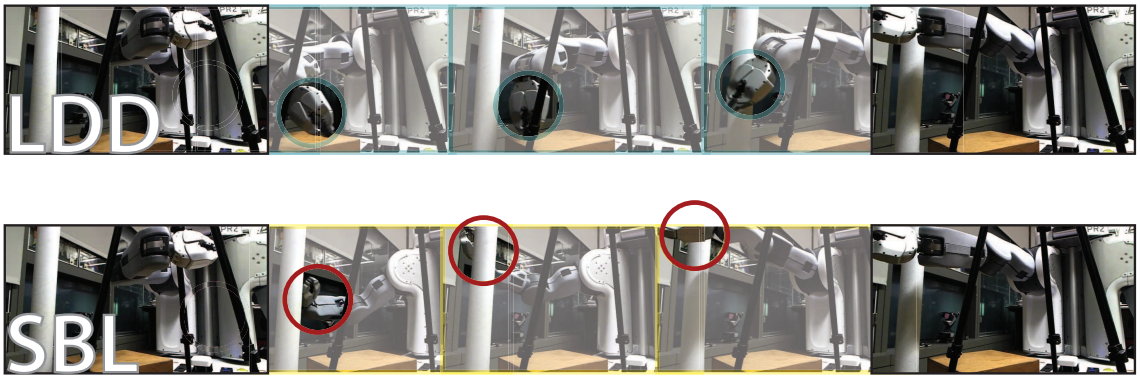
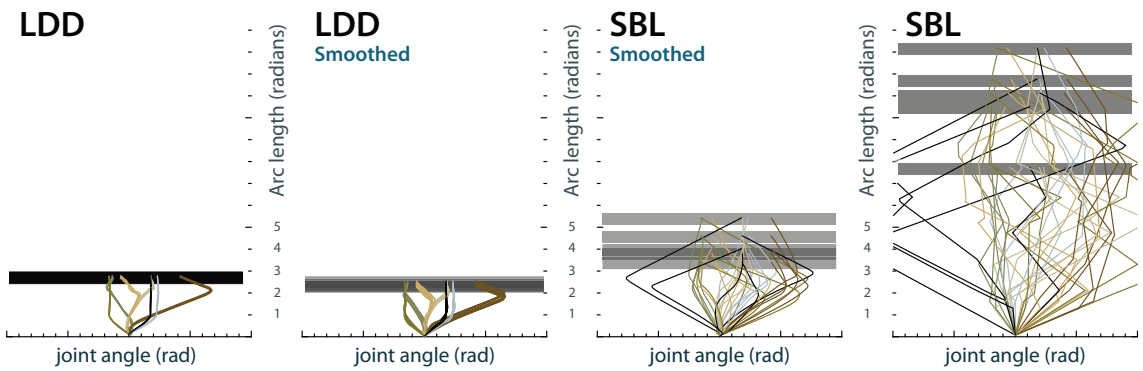
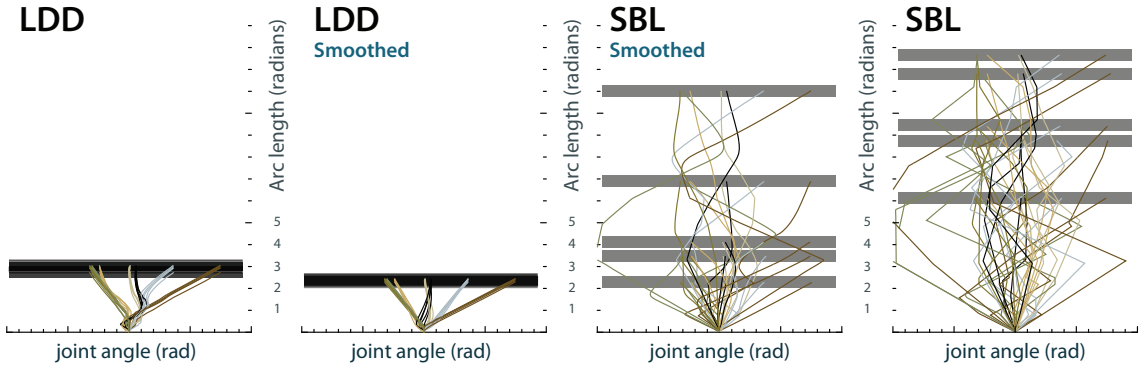


Figure 17: Video stills from *poles* experiment comparing worst-case performance of LDD and SBL (both with smoothing). End-effector is highlighted in intermediate frames. SBL path takes a long detour behind and over the pole on the left, whereas LDD takes a much more direct path.



(a) *Windows* experiment



(b) *Poles* experiment

Figure 18: Experimental comparison of repeatability and quality of trajectories obtained with LDD and SBL, with and without post-smoothing, for each of two scenarios. Plots are interpreted as described in Fig. 14, albeit rotated 90 degrees. Although smoothing helps to narrow the gap somewhat, LDD produces trajectories that are much more smooth and consistent across trials. Post-smoothing has little effect on the LDD trajectories, presumably because they are nearly optimal in the first place.

provided fairly little benefit in terms of decreasing arc length (vertical axis), indicating that these solutions were probably nearly optimal to begin with—this is again despite the fact that arc length was not minimized exactly due to a number of factors. As expected, the SBL trajectories were chaotic, inconsistent, and long across trials before smoothing. Smoothing these sometimes yielded dramatic decreases in arc length, but results were generally unpredictable. In the windows experiment, the unsmoothed LDD output always surpassed even the smoothed output of SBL. This was mostly true as well for the poles experiment, save for one trial in which smoothed SBL performed just as well as smoothed LDD. Finally, note that the consistency of SBL between trials was not obviously improved by smoothing.

6.4 Analysis

In this section, we briefly examine two issues relating to the performance of LDD. First, we give a simple result illustrating how concrete approximation guarantees might be obtained for LDD. We then discuss the issue of local minima, demonstrating a simple case where a local minimum might lead to a poor solution.

6.4.1 Approximation guarantee

As previously mentioned, LDD is guaranteed to find the globally optimal path (up to discretization) in one step provided that $C(x) = C(W^1 W^{1T} x)$, $\forall x$. In practice, this condition will rarely be met with a basis W^1 containing only a few columns, which is the only case in which we can practically apply DP to find a solution. We therefore consider a particular relaxation of the condition that $C(x) = C(W^1 W^{1T} x)$, $\forall x$ and show that in this case, LDD produces results of provable suboptimality, given that we can additionally lower-bound the cost function as to be described shortly.

We first consider a variant of LDD—denoted LDD'—that approximates the cost function C by a *compressed* cost function C' such that $C(x) \approx C'(x) = C(W^1 W^{1T})$ and $C'(x) \geq C(x)$, $\forall x$ (this variant is essentially SLASHDP). The following result then applies.

Theorem 6.1. *Suppose LDD' is applied to a problem with cost function C, and a compressed cost C' is obtained such that $C'(x) \geq C(x)$ and $C'(x) - C(x) \leq \delta$, $\forall x$. Let $J\{x; C\} = \int \|\dot{x}\| C(x) dt$ denote the cost functional evaluated on path x using cost function C, let $x^* = \arg\min_x J\{x; C\}$, and let $x'^* = \arg\min_x J\{x; C'\}$. Then, assuming $C(x) \geq 1$, $\forall x$,*

$$\frac{J\{x'^*; C\} - J\{x^*; C\}}{J\{x^*; C\}} \leq \delta. \tag{27}$$

Proof. We first have that for any path $x(t)$,

$$\begin{aligned} J\{x; C'\} - J\{x; C\} &= \int (C'(x) - C(x)) \|\dot{x}\| dt \\ &\leq \delta \int \|\dot{x}\| dt. \end{aligned} \tag{28}$$

We also observe that

$$J\{x^*; C\} \leq J\{x'^*; C\} \leq J\{x'^*; C'\} \leq J\{x^*; C'\}. \tag{29}$$

Therefore,

$$\begin{aligned} J\{x'^*; C\} - J\{x^*; C\} &\leq J\{x^*; C'\} - J\{x^*; C\} \\ &\leq \delta J\{x^*; C\}, \end{aligned} \tag{30}$$

which is equivalent to the desired result. □

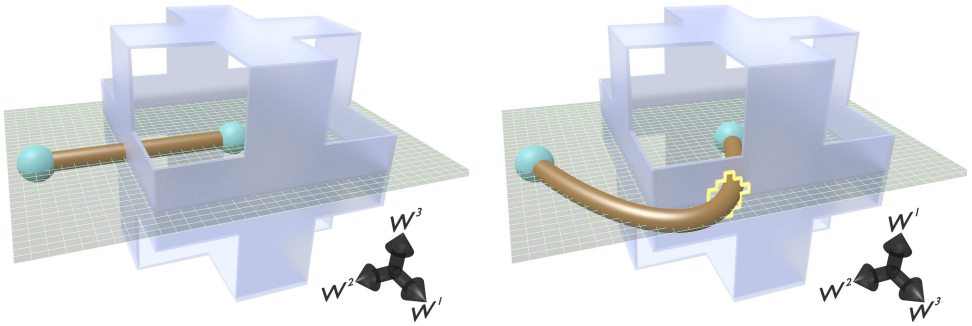
The δ in the above theorem can be thought of a surrogate for how well the low dimensional approximation holds. The result implies that the solution returned by LDD' is boundedly suboptimal in terms of δ, as expected. It is simple to show that this result also holds for LDD, since the LDD path is guaranteed to be no more costly than the LDD' path.

6.4.2 Local minima

LDD therefore begins with an solution that is optimal with respect to an approximated cost and, as previously mentioned, improves upon it monotonically (neglecting discretization effects). However, LDD is vulnerable to local minima, a simple example of which is given here.

Figure 19(a) illustrates a local minimum that occurs in a three-dimensional problem if two-dimensional search submanifolds are employed. The figure illustrates a cage-like obstacle created by carving cubes out of a large hollow cube at the corners. The endpoints are illustrated as spheres, and the initial solution is illustrated as a tube drawn between them. In the first iteration, the plane illustrated by the grid is searched for an improved solution; however, no progress is possible, as no collision-free path exists between the endpoints lying in the plane. The same is true for all subsequent iterations, by symmetry. A local minimum is therefore immediately encountered. Figure 19(b) shows how this local minimum is fairly unstable with respect to changes in the cost function. In this case, a hole in the cage at the level of any search plane ensures that LDD is able to find a better, collision-free local minimum.

In practice, minima of the sort encountered in Fig. 19(a) could be escaped by introducing a small amount of random noise either into the path or the basis selection, which would likely result in a submanifold containing collision-free paths. Alternatively, raising the dimension of the searched submanifolds upon finding a local minimum may prove sufficient to escape such minima. In the pictured example, raising the dimension of the searched submanifolds by one would cure the problem, at the expense of searching over the full three dimensional space.



(a) Poor local minimum (b) Better local minimum

Figure 19: Illustration of LDD local minima with respect to two-dimensional search submanifolds. Fig. 19(a) shows a poor local minimum for a certain problem, while Fig. 19(b) shows a much better local minimum obtained in a slightly modified problem (note hole in wall). Dark arrows show learned basis.

7 Conclusions

We have introduced a novel class of methods to solve the optimal holonomic motion planning problem based on learning and exploiting low dimensional structure. Problems possessing exact low dimensional structure may be solved exactly (up to discretization) with a complexity depending on the dimension of a salient subspace, instead of the dimension of the configuration space. We can then develop methods for general problems by solving subproblems to which this insight may be applied.

Two closely related methods, SLASHDP and LDD, were given to implement these ideas. SLASHDP was derived from the idea that if the cost function can be compressed by its representation in a low dimensional subspace, then we can exactly compute a compressed value function for this problem. LDD was designed to generalize SLASHDP in most cases and automatically derives a cascade of planning problems, chosen intelligently so as to avoid becoming stuck prematurely in local minima.

Experimental results demonstrated that these methods were able to find high-quality solutions in a variety of scenarios, including a real-world arm planning scenario. We found that SLASHDP was able to solve a 144-dimensional planar arm problem by performing dynamic programming in a 4-dimensional space. LDD reliably found higher-quality solutions in a real-world arm planning problem than traditional methods without excessive computational cost.

We believe that there results demonstrate that there is merit in thinking of the notion of cost dimensionality as a useful surrogate for the intrinsic hardness of a particular motion planning problem. Although further work needs to be done to formalize and validate this concept, we believe that it will provide fertile ground for future developments in optimal motion planning.

8 Funding

Partial funding for this work was provided by ONR Project N00014-09-1-1051.

A Proof of Theorem 4.1

The constructive proof of Theorem 4.1 given here is a straightforward application of the calculus of variations [Arnold, 1989]. First, the assumption of the theorem implies that in some coordinates, the cost function C only depends on the first d coordinates. We therefore assume w.l.o.g. that the coordinates x_i are such that C depends only on the first d of these coordinates; i.e., $\partial C/\partial x_i = 0 \ \forall i > d, \forall x$. We will refer to coordinates on which the cost function does not depend as *cyclic* coordinates, to borrow a physics term.

We then define the Lagrangian

$$L(x, \dot{x}) = \|\dot{x}\| C(x)$$

and apply the Euler-Lagrange equations, yielding

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{x}_i} - \frac{\partial L}{\partial x_i} = \frac{d}{dt}\left[\frac{\dot{x}_i}{\|\dot{x}\|} C(x)\right] - \|\dot{x}\|\frac{\partial C}{\partial x_i} = 0.$$

For any cyclic coordinate, we can substitute $\partial C/\partial x_i = 0$, implying that $\forall t$, and for some yet-unknown constants k_i,

$$\frac{\dot{x}_i}{\|\dot{x}\|} C(x) = k_i. \tag{31}$$

We note without detailed proof that we can assume that $\|\dot{x}\| = \alpha$, where α is some irrelevant constant, since (2) can be shown to be invariant with respect to time-reparameterizations of $x(t)$ (i.e., moving faster or slower along some part of the path never yields any change in cost). (31) then yields $N - d$ independent, separable ODEs for each cyclic coordinate. Integration of these produces

$$\begin{aligned} x_i(t) &= \alpha k_i \int_0^t \frac{1}{C(x(t))} dt \\ &= \alpha k_i F(t), \end{aligned} \tag{32}$$

defining $F(t)$ in the last step, and where we have assumed w.l.o.g. that $x(0) = 0$.

We then write the path as a linear combination of the standard basis vectors, e_i:

$$x(t) = \sum_i x_i(t) e_i$$

and substitute (32) into this equation, which produces

$$x(t) = \sum_{i=1}^{d} x_i(t) e_i + \sum_{i=d+1}^{N} \alpha k_i F(t) e_i.$$

We now solve for the k_i in this expression to express the last basis vector in terms of the problem data. The k_i can be computed from the final conditions; i.e., $x_i(1) = \alpha k_i F(1)$. Substitution of this expression and simplification then yields

$$x(t) = \sum_{i=1}^{d} x_i(t) e_i + \frac{F(t)}{F(1)} \sum_{i=d+1}^{N} x_i(1) e_i.$$

By expressing the basis e_i in terms of the basis w_i, we can obtain the desired expression, with $(x_1(t), \ldots, x_d(t))$ becoming $a(t)$, and $F(t)/F(1)$ becoming $s(t)$.

References

[omp, 2010] (2010). The Open Motion Planning Library (OMPL).

[Amato and Wu, 1996] Amato, N. M. and Wu, Y. (1996). A randomized roadmap method for path and manipulation planning. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 113–120.

[Arnold, 1989] Arnold, V. I. (1989). *Mathematical Methods of Classical Mechanics*. Springer-Verlag.

[Barraquand and Ferbach, 1994] Barraquand, J. and Ferbach, P. (1994). Path planning through variational dynamic programming. In *IEEE International Conference on Robotics and Automation*, pages 1839–1846. IEEE.

[Berenson et al., 2011] Berenson, D., Simeon, T., and Srinivasa, S. (2011). Addressing cost-space chasms for manipulation planning. In *ICRA*.

[Berenson et al., 2009] Berenson, D., Srinivasa, S., Ferguson, D., Collet, A., and Kuffner, J. (2009). Manipulation planning with workspace goal regions. In *ICRA*.

[Bertsekas, 2005] Bertsekas, D. (2005). *Dynamic Programming and Optimal Control: Volume I*. Athena Scientific.

[Brock and Kavraki, 2001] Brock, O. and Kavraki, L. (2001). Decomposition-based motion planning: a framework for real-time motion planning in high-dimensional configuration spaces. In *ICRA*.

[Bulitko et al., 2005] Bulitko, V., Sturtevant, N., and Kazakevich, M. (2005). Speeding up learning in real-time search via automatic state abstraction. In *Proceedings of the National Conference on Artificial Intelligence*.

[Cohen et al., 2011] Cohen, B., Subramanian, G., Chitta, S., and Likhachev, M. (2011). Planning for manipulation with adaptive motion primitives. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

[Dalibard and Laumond, 2009] Dalibard, S. and Laumond, J. P. (2009). Control of probabilistic diffusion in motion planning. *Algorithmic Foundation of Robotics VIII*.

[Diankov and Kuffner, 2007] Diankov, R. and Kuffner, J. (2007). Randomized statistical path planning. In *IROS*, pages 1–6. IEEE.

[Diankov et al., 2008] Diankov, R., Ratliff, N., Ferguson, D., Srinivasa, S., and Kuffner, J. (2008). Bispace planning: Concurrent multi-space exploration. *RSS*.

[Do Carmo, 1976] Do Carmo, M. (1976). *Differential geometry of curves and surfaces*. Prentice-Hall Englewood Cliffs, NJ.

[Ferguson and Stentz, 2006] Ferguson, D. and Stentz, A. (2006). Anytime RRTs. In *IROS*. IEEE.

[Geraerts and Overmars, 2004] Geraerts, R. and Overmars, M. (2004). Clearance based path optimization for motion planning. In *ICRA*, pages 2386–2392.

[Geraerts and Overmars, 2007] Geraerts, R. and Overmars, M. H. (2007). Creating high-quality paths for motion planning. *The International Journal of Robotics Research*, 26(8):845–863.

[Hauser et al., 2008] Hauser, K., Bretl, T., Latombe, J., and Wilcox, B. (2008). Motion planning for a six-legged lunar robot. *Algorithmic Foundation of Robotics VII*, pages 301–316.

[Hecht, 2001] Hecht, E. (2001). *Optics*. Addison Wesley.

[Holte et al., 1996] Holte, R., Perez, M., Zimmer, R., and MacDonald, A. (1996). Hierarchical A*: Searching abstraction hierarchies efficiently. In *Proceedings of the National Conference on Artificial Intelligence*, pages 530–535.

[Hsu et al., 1999] Hsu, D., Latombe, J.-C., and Motwani, R. (1999). Path planning in expansive configuration spaces. *International Journal Computational Geometry & Applications*, 4:495–512.

[Jacobson and Mayne, 1970] Jacobson, D. and Mayne, D. (1970). *Differential dynamic programming*. Elsevier Sci. Publ.

[Karaman and Frazzoli, 2010] Karaman, S. and Frazzoli, E. (2010). Incremental sampling-based algorithms for optimal motion planning. In *RSS*, Zaragoza, Spain.

[Karaman and Frazzoli, 2011] Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894.

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

[Kolter et al., 2008] Kolter, J., Rodgers, M., and Ng, A. (2008). A control architecture for quadruped locomotion over rough terrain. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 811–818. Ieee.

[Kuffner and LaValle, 2000] Kuffner, J. and LaValle, S. (2000). RRT-connect: An efficient approach to single-query path planning. In *ICRA*.

[LaValle, 2006] LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press, Cambridge, U.K. Available at http://planning.cs.uiuc.edu/.

[LaValle and Kuffner, 1999] LaValle, S. M. and Kuffner, J. J. (1999). Randomized kinodynamic planning. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 473–479.

[Li and Todorov, 2004] Li, W. and Todorov, E. (2004). Iterative linear-quadratic regulator design for nonlinear biological movement systems. In *Proceedings of the First International Conference on Informatics in Control, Automation, and Robotics*, pages 222–229. Citeseer.

[Likhachev et al., 2004] Likhachev, M., Gordon, G., and Thrun, S. (2004). ARA*: Anytime A* with provable bounds on sub-optimality. In *NIPS*.

[Likhachev and Stentz, 2008] Likhachev, M. and Stentz, A. (2008). R* search. In *AAAI*.

[Morimoto et al., 2003] Morimoto, J., Zeglin, G., and Atkeson, C. (2003). Minimax differential dynamic programming: application to a biped walking robot. In *IROS*.

[Poupart and Boutilier, 2003] Poupart, P. and Boutilier, C. (2003). Value-directed compression of POMDPs. *Advances in Neural Information Processing Systems*, pages 1579–1586.

[Quinlan and Khatib, 1993] Quinlan, S. and Khatib, O. (1993). Elastic bands: Connecting path planning and control. In *In Proceedings of the International Conference on Robotics and Automation*, pages 802–807.

[Ratliff et al., 2009] Ratliff, N., Zucker, M., Bagnell, J., and Srinivasa, S. (2009). CHOMP: Gradient optimization techniques for efficient motion planning. In *ICRA*.

[Rebula et al., 2007] Rebula, J., Neuhaus, P., Bonnlander, B., Johnson, M., and Pratt, J. (2007). A controller for the littledog quadruped walking on rough terrain. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1467–1473. IEEE.

[Roy and Gordon, 2003] Roy, N. and Gordon, G. (2003). Exponential family PCA for belief compression in POMDPs. *Advances in Neural Information Processing Systems*, pages 1667–1674.

[Sánchez and Latombe, 2001] Sánchez, G. and Latombe, J.-C. (2001). A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *Proceedings International Symposium on Robotics Research*.

[Sethian, 1996] Sethian, J. A. (1996). A fast marching level set method for monotonically advancing fronts. *PNAS*, 93(4):1591–1595.

[Shkolnik and Tedrake, 2009] Shkolnik, A. and Tedrake, R. (2009). Path planning in 1000+ dimensions using a task-space Voronoi bias. In *ICRA*.

[Sucan and Kavraki, 2008] Sucan, I. A. and Kavraki, L. E. (2008). Kinodynamic motion planning by interior-exterior cell exploration. In *WAFR*.

[Todorov and Li, 2005] Todorov, E. and Li, W. (2005). A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference, 2005. Proceedings of the 2005*, pages 300–306. IEEE.

[Vernaza, 2011] Vernaza, P. (2011). *Efficient learning and inference for high-dimensional Lagrangian systems*. PhD thesis, University of Pennsylvania.

[Vernaza and Lee, 2011a] Vernaza, P. and Lee, D. D. (2011a). Efficient dynamic programming for high-dimensional, optimal motion planning by spectral learning of approximate value function symmetries. In *ICRA*.

[Vernaza and Lee, 2011b] Vernaza, P. and Lee, D. D. (2011b). Learning dimensional descent for optimal motion planning in high-dimensional spaces. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

[Vernaza et al., 2009] Vernaza, P., Likhachev, M., Bhattacharya, S., Chitta, S., Kushleyev, A., and Lee, D. D. (2009). Search-based planning for a legged robot over rough terrain. In *ICRA*.

[Zucker et al., 2010] Zucker, M., Bagnell, J., Atkeson, C., and Kuffner, J. (2010). An optimization approach to rough terrain locomotion. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3589–3595. IEEE.