Efficient dynamic programming for high-dimensional, optimal motion planning by spectral learning of approximate value function symmetries

Paul Vernaza and Daniel D. Lee

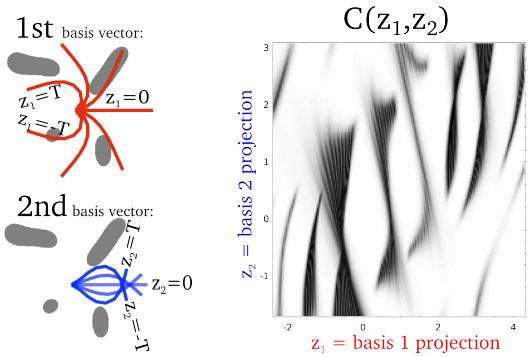
***Abstract*— We demonstrate how to find high-quality motion plans for high-dimensional holonomic systems efficiently using dynamic programming in a learned subspace of vastly reduced dimension. Our approach (SLASHDP) learns the low-dimensional cost structure of an optimal control problem via an efficient spectral method. This structure results in a symmetric value function that serves as a an efficiently-computable surrogate for the true value function. High-quality feedback motion plans can then be obtained from the symmetric value function. Experimental results show that SLASHDP yields higher-quality plans than can be obtained by post-processing plans generated by a sampling-based motion planner, and with less computational effort for very high-dimensional problems. We demonstrate high-quality dynamic programming plans for an arm planning problem of up to 144 dimensions without using any domain-specific knowledge aside from that learned automatically by SLASHDP. Positive results are also shown for a high-dimensional deformable robot planning problem.**

I. Introduction

We consider in this work the problem of kinematic motion planning in high-dimensional spaces with holonomic constraints [1]. As is well known, this is generally considered to be a computationally intractable problem in the most general case. Fortunately, this does not rule out the possibility that special cases of this problem can be solved efficiently, nor does it rule out the existence of practical algorithms to solve many interesting motion planning problems. The central concept of our work is that we can efficiently solve a certain special case of this problem. This solution can then be leveraged to approximately solve more interesting problems.

To be more specific, we consider an optimal-control-type version of the motion planning problem, and look for structure in the cost function of this formulation. Namely, if the cost function is *low-dimensional* (i.e., depends on only a few coordinates in some basis), we can show that the associated motion planning problem can be solved easily and efficiently. The challenge, then—and the main topic of this manuscript—is the automatic discovery of such structure. We refer to our approach to this problem as Spectral Learning of Approximate Symmetries for High-dimensional Dynamic Programming (SLASHDP), for reasons that should hopefully become apparent soon.

An informal example will serve to elucidate these ideas. Figure 1 depicts a simple arm planning problem in the presence of obstacles. Our goal will be to find a small set of 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 approximate the true cost function by a low-dimensional cost function written as function only of the basis coordinates. Fig. 1(a) shows the best two basis motions learned by SLASHDP for the particular obstacle field depicted. The first motion performs a curling motion that emphasizes motion of the joints near the base. As was 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.



(a) Learned basis (b) Learned subspace cost

Fig. 1. Visualization of learned basis vectors and cost as a function of basis coordinates for an arm planning problem. Basis vectors are illustrated in 1(a). Each line shows a configuration $z_i u_i$, for $i \in \{1, 2\}$ and varying values of z_i, superimposed on an obstacle map from which the cost function is derived. Sampled cost function in (z_1, z_2) coordinates is displayed in 1(b).

Letting z_1, z_2 denote the projections of the state onto these basis vectors, we now discretize z_1, z_2 space and numerically estimate the cost as a function of just these coordinates. The result is shown in Figure 1(b). We can think of this figure as a visualization of obstacle proximity in some *generalized configuration space* that is a rotation and projection away from our original configuration space. We can see from the figure that if we start at the origin and move forward along the first basis vector, the arm will intersect two obstacles, visible as tall shapes stretched out in the up-down direction. However, these obstacles have limited extent in the z_2 direction. It is easy to see how we could possibly navigate around them by moving up or down in the z_2 direction as we move forward in z_1. This is precisely the kind of behavior that SLASHDP will exhibit when we generate a high-dimensional feedback plan for this low-dimensional cost function.

P. Vernaza and D. D. Lee are with the GRASP Lab, University of Pennsylvania, 3330 Walnut St., Philadelphia, PA 19104 {vernaza,ddlee}@seas.upenn.edu

A. Previous work

Numerous techniques for high-dimensional motion planning have arisen in the last two decades or so. The most popular and practically useful of these are probably the class of sampling-based motion planning algorithms, such as RRT [2], PRM [3][4], SBL [5], and EST [6]. Also notable are certain modern variants of A*, such as ARA* [7] and R* [8].

Our method is related to such deterministic search methods in that we eventually perform a deterministic search to find a plan. In our case, this amounts to simple dynamic programming, though we could easily imagine variants of our work that employ more sophisticated deterministic search methods. On the other hand, our work differs from all of the deterministic and randomized search methods mentioned above in that the search algorithm itself is of secondary importance to SLASHDP. The critical aspect of our work is the learning of low-dimensional structure, which we feel is a much under-appreciated topic.

Perhaps the main reason why this might be so is the fact that all of the above algorithms introduce discretization immediately—after which the concept of geometric dimension becomes lost. We reason about concepts such as the dimensionality of the cost function and its associated optimal paths *before* performing any discretization, which allows us to use our finite samples much more efficiently. Although there is some existing literature on learning heuristics for search ([9], [10], [11]), and other work on exploiting different sorts of low-dimensional structures in motion planning ([12], [13]), we believe that none of these approaches is able to exploit the kind of structure we examine, for precisely this reason. One could argue that the more limited assumptions of these methods is an advantage; however, if one is trying to solve the kind of motion planning problem described here, we would argue that any method that does not exploit low-dimensional cost structure, is immediately at a disadvantage.

II. Preliminaries

We begin with some background information about our particular problem and its generic solution via dynamic programming.

In this work, we address the feedback optimal motion planning problem with holonomic constraints. We further assume that such constraints (such as obstacle penetration constraints) can be adequately modeled by a suitable strictly positive cost function $C(x)$ that penalizes the state x appropriately, rather than strictly enforcing these constraints. $C(x)$ may also encode other arbitrary penalties.

The goal of our motion planning problem is then to find an optimal path $x(t)$ that minimizes the cost under $C(\cdot)$ of moving from some point to the origin. Furthermore, we would like to generate an entire family of such solutions by means of a feedback control law that for any state x, guides the system along an optimal path to the origin, for a total cost of $V(x)$. These concepts are summarized in the following optimization problem.

Definition 2.1 (Feedback optimal motion planning):

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

This indicates that for every state x in some region of state space, we would like to compute the optimal value $V(x)$ by searching over the set of all paths with one endpoint at the origin and the other at x. In typical dynamic programming fashion, the optimal control law $x \to \dot{x}$ can be computed as the negative gradient of $V(x)$:

$$\dot{x} = -\nabla V(x) \tag{2}$$

As is well known, dynamic programming—typically in the form of the Fast Marching Method [14]—can be employed to compute the value function on a sampled lattice in time nearly linear in the size of the lattice. Unfortunately, a straightforward discretization of a high-dimensional state space yields a number of samples that scales exponentially in the dimension, making straightforward calculation of the value function impossible.

A. Symmetry of the value function

In this this work, however, we assume that the cost function has some low-dimensional structure which will allow us to compute the value function efficiently. To elaborate, we define the *dimension* of a cost function to be the number of coordinates on which the cost function depends, or to be more precise:

Definition 2.2: A (differentiable) cost function $C(x)$ *depends* on a coordinate x_i iff. there exists an x such that $\partial C(x)/\partial x_i \neq 0$. The *dimension* of a cost function is the number of coordinates on which it depends.

Our core result is that the value function of a motion planning problem with a low-dimensional cost function, is symmetric in a way that allows us to efficiently compute it, as described in the following theorem:

Theorem 2.1: Let $C(x_1, \ldots, x_N) : \mathbb{R}^N \to \mathbb{R}$ be a d-dimensional cost function, let $\hat{C}(x_1, \ldots, x_{d+1}) : \mathbb{R}^{d+1} \to \mathbb{R}$ denote the restriction of C to the first $d+1$ coordinates, and let $V(x_1, \ldots, x_N) : \mathbb{R}^N \to \mathbb{R}$ and $\hat{V}(x_1, \ldots, x_{d+1}) : \mathbb{R}^{d+1} \to \mathbb{R}$ denote the value functions associated with the corresponding motion planning problems. Then

$$V(x_1, \ldots, x_N) = \hat{V}(x_1, \ldots, x_d, \sqrt{x_{d+1}^2 + \cdots + x_N^2})$$

Note that this restriction is well-defined because the cost function does not depend on the other coordinates. The preceding theorem implies that if we wish to compute the value function of a motion planning problem with a d-dimensional cost function, we need only compute the value function of a $d+1$-dimensional motion planning problem, *regardless of the dimension of the state space of the original problem*. Figure 2 illustrates this for a simple case where the cost function is one-dimensional in an ambient three-dimensional space.

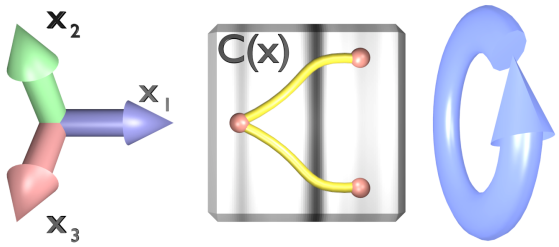


Fig. 2. Illustration of symmetry of value function for a shortest-path problem in three dimensions with a cost function that depends only on the x_1 coordinate. The corresponding value function is symmetric about the x_1 axis. Symmetric optimal paths are shown in yellow.

A useful corollary to this theorem is the following, which states that optimal paths of low-dimensional cost functions lie in low-dimensional subspaces.

Corollary 2.2: Consider a motion planning problem of the form of Definition 2.2 having a d-dimensional cost function. Any optimal path $y^*(t)$ of such a problem with nonzero endpoint x, lies completely in the $d+1$-dimensional subspace spanned by the first d coordinates and x. i.e., there exist functions $y_a(t) : \mathbb{R} \to \mathbb{R}^d$ and $y_c(t) : \mathbb{R} \to \mathbb{R}$ such that $y_c(t) \in [0, 1]\ \forall t$ and

$$y^*(t) = I_{N\times d} y_a(t) + y_c(t)x,\ \forall t$$

where $I_{N\times d}$ is the identity matrix augmented with rows of zeros.

Both Theorem 2.1 and its corollary can be proved in a straightforward way via the calculus of variations. We omit the proofs here for lack of space, instead focusing on applications of these results.

III. Method

We have so far described how a low-dimensional cost function is useful, but we have not described how one might be obtained. Certain problems may have a-priori structure that might be exploited manually to yield a low-dimensional cost function. In this work, however, we consider the idea of learning such structure automatically for a given problem.

Our approach to the learning problem is quite simple. To summarize, we estimate the matrix of second moments of the gradient of the cost function by random sampling. The top d eigenvectors of this matrix then form a basis in which the cost is approximately low-dimensional. We then discretize this reduced space and deterministically sample the cost in these coordinates. This discrete, low-dimensional cost function is finally used to calculate the value function via dynamic programming.

A. Learning a cost basis

The first critical step in SLASHDP is to estimate a basis in which the cost is approximately low-dimensional. That is, suppose our problem is described in some original coordinates x, and we have a cost function

$$C(x) = f(x_1, x_2, \ldots, x_N)$$

i.e., C is a function of all N variables. We then seek a rotation matrix U and new coordinates z such that $x = Uz$ and

$$C(x) = C(Uz) = g(z_1, z_2, \ldots, z_d)$$

where $d \ll N$.

In practice, we need only to find a few columns of U. To find these, we note that $C(Uz)$ does not depend on some z_i iff. $\partial C/\partial z_i = 0$ for all z. Therefore, it would make sense to choose z_1 such that $\|\partial C/\partial z_1\|$ is very large in expectation. Since (letting u_i be the ith column of U)

$$\partial C/\partial z_1 = \nabla C^T u_1$$

we can choose u_1 to satisfy this goal by solving the following optimization problem:

$$u_1 = \arg\max_{\|u\|=1} \mathbb{E}_x u^T \nabla C(x) \nabla C(x)^T u \tag{3}$$

We refer to $M := \mathbb{E}_x \nabla C(x) \nabla C(x)^T$ as the *matrix of second moments* of the cost gradient. The well-known solution to the above optimization yields that u_1 should be chosen to be the eigenvector corresponding to the largest eigenvalue of M. Subsequent columns $u_2, \ldots, u_d$ of U can then be chosen as the eigenvectors corresponding to eigenvalues in order of descending magnitude.

Of course, since M is not known a-priori, it must be estimated by sampling. Although sampling in high-dimensions is generally difficult, intuition suggests that if the spectrum of M is sufficiently tapered, it should not be difficult to estimate the eigenvectors corresponding to its largest eigenvalues. As an extreme example of this, we might consider the case where M has only one nonzero eigenvalue: drawing any single sample of the cost gradient will perfectly recover the eigenvector corresponding to the nonzero eigenvalue, but we will never draw a sample corresponding to any other eigenvalue. It is therefore logical to expect that eigenvectors corresponding to large eigenvalues are easy to estimate, while those corresponding to small eigenvalues are difficult to estimate; fortunately, we have no need to estimate the latter.

B. Estimating the transformed cost function

Given a basis U, we now wish to estimate the cost function in the reduced coordinates, $C(Uz)$. We 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 $C(Uz[i])$. This is very simple in the ideal case; since $C(Uz)$ truly only depends on $d \ll N$ z coordinates, we can take $C(Uz[i])$ to be the value of $C(x)$ for any x such that $x = Uz$; all such x will have the same cost.

In the likely event that $C(\cdot)$ is not perfectly d-dimensional, different states that project to the same z coordinates will 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 $C(Uz)$

to the mean, max, or min of all such values. However, we can do much better given a reasonable assumption.

Let us assume for now that we are only interested in finding the optimal path to the goal for a particular *query* state x_q. In this case, we know by Corollary 2.2 that the optimal path will be contained in a certain known subspace. When sampling to obtain $C(Uz)$, we should therefore only choose samples that lie in this subspace.

Concretely, suppose we wish to find a sample $x[i]$ that projects to some coordinates $z[i]$. We can write any such $x[i]$ as

$$x[i] = Uz[i] + (I - UU^T)y$$

for some y. If we want $x[i]$ to lie in the optimal subspace pertaining to x_q, then Corollary 2.2 implies that we must choose y such that

$$y = x_q s$$

for some scalar $s \in [0, 1]$ to be determined. We have therefore reduced our problem from having to sample in the $N-d$-dimensional null space of U to having only to sample in a one-dimensional subspace.

In practice, we have found that sampling s is actually unnecessary; instead, we deterministically choose s by the following heuristic. For a given sample $z[i]$, and writing $z_q = U^T x_q$, we choose

$$s = \frac{\|z[i]\|}{\|z_q\| + \|z[i] - z_q\|}$$

. This ensures that $x[i] = 0$ when $z[i] = 0$, $x[i] = x_q$ when $z[i] = z[q]$, and a smooth interpolation elsewhere.

We note that these simplifications depend on the choice of a specific query configuration. Therefore, one might question whether a feedback policy obtained with this assumption would still be valid for other query configurations. Though we will not delve deeply into this issue, it seems reasonable to expect that the resulting feedback policy will be valid for at least a neighborhood of states surrounding the query optimal path. Our experimental results support this intuition.

C. Planning a path

After learning a d-dimensional basis in which the cost is approximately low-dimensional and discretely sampling the cost in these coordinates, we can use the Fast Marching Method to compute a $d+1$-dimensional value function from which we can derive a high-dimensional value function by invoking Theorem 2.1. This value function is a symmetric approximation of the problem's true value function, which cannot be computed tractably. Since we have learned a basis such that most of the variation of the cost is captured in this basis, we anticipate that the resulting symmetric approximate value function will be a good approximation to the true one.

Given the high-dimensional, symmetric, approximate value function obtained in this way, we can efficiently compute the corresponding optimal paths by integrating (2). In practice, however, we compute these paths by the equivalent method of integrating (2) with respect to the $d + 1$-dimensional value function and lifting these paths to the original high-dimensional space by means of Corollary 2.2.

D. Locally low-dimensional paths

The method we have described so far is useful for problems where the original coordinates are a rotation away from a new set of coordinates in which the cost is approximately low-dimensional. However, for some problems, this may only be true locally. For such problems, we suggest the following approach: given an initial path, randomly choose two points on this path, and find a new path between them via SLASHDP. If the cost of the found path is lower than the original segment, replace the old segment with the new one; otherwise, keep the original segment. Repeat as many times as desired.

For each such iteration, we can focus our learning efforts on a smaller search volume surrounding the path endpoints. It is our hope that the cost function in this smaller volume can be adequately represented as a function of a small number of basis projections. If this does hold, we expect to be able to find a nearly optimal path interpolating these endpoints.

This method can be seen as a generalization of the *straight-line shortcut heuristic* [15] 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 of our method that arises when a 0-dimensional (constant) approximation of the cost function is used. SLASHDP, however, can employ higher-dimensional cost approximations, resulting in *higher-dimensional, nonlinear* shortcuts—if we use a d-dimensional cost approximation, we can search over a space of paths spanning a $d + 1$-dimensional space.

E. 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 cost approximation using k samples per dimension, sampling the cost function takes time $O(k^d)$, and dynamic programming takes time $O(k^{d+1}(d+1)\log k)$, due to the need to augment the cost function with an additional dimension in order to be able to compute the symmetric value function. 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 sampling steps are embarrasingly 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 (say, for instance, using a GPU).

IV. Results

We have applied the methods described so far to two problems: planning for a robot arm with many degrees of freedom and planning for a deformable robot. Both are challenging high-dimensional motion planning problems.

A. Planning for a robot arm

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

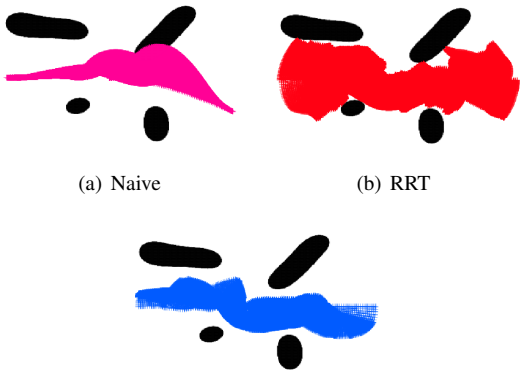
$$J[q] = \int_0^1 (1 + e^{-(d_{obs}(q(t)) - d_0)/\bar{d}}) \|\dot{q}\| dt$$

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.

We compared the performance of SLASHDP on this scenario to two others: a bidirectional RRT [2], and a naive method based on a low-dimensional projection. The naive method calculates a two-dimensional feedback plan for the end-effector that optimizes a similar objective to the one above, but only considering the end-effector coordinates independently of the rest of the arm. It then lifts this plan to joint space by mapping the feedback controller's desired end-effector velocity to joint velocities via the pseudoinverse of the Jacobian. The RRT consists of a standard bidirectional RRT using linear interpolation as a local planning method and Euclidean distance as a distance metric.

Qualitative results for this experiment are shown in Figure 3. Each subfigure shows the set of points swept out by a 36-dimensional arm as it travels along the solution trajectory obtained with each method. The naive method produces a smooth solution, but it collides with an obstacle, since the geometry of the arm was not taken into consideration during construction of the low-dimensional path. 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.

These qualitative observations are supported by the quantitative results in 4. In this experiment, we compared the three methods as we scaled the dimensionality of the arm up to 144 joints. The cost of an initial solution found via each method is shown in Figure 4(a) (note log scale). As expected, the naive solution always had a higher cost than the other methods, since it consistently collided with obstacles. The RRT produced a lower cost due to its lack of collision, but the length of these solutions coupled with their occasional proximity to obstacles still caused them to have a relatively high cost. SLASHDP produced solutions that were consistently on the order of 100 to 200 times less costly than the RRT for very high-dimensional problems. Furthermore, the cost of the SLASHDP solutions actually decreased monotonically as we increased the dimensionality of the problem while the other methods exhibited either a stable or generally increasing trend. This implies that SLASHDP, instead of being confused by the extra dimensions, was able to exploit the extra degrees of freedom to further decrease the cost.



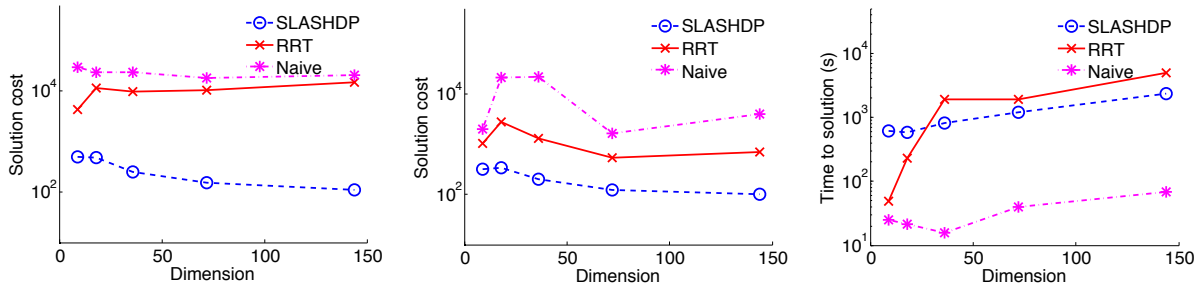
(a) Naive (b) RRT

(c) SLASHDP

Fig. 3. 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. Naive method produces a smooth solution, but it collides with an obstacle. 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.

We then post-processed each of these solutions using a local smoothing method (in particular, an elastic band [16]), not stopping until a local optimum was found. All of the methods benefitted from this step, though to varying degrees. Post-processing decreased the RRT and naive costs by a factor of 5-20–still not enough to surpass the quality of the initial plans generated by SLASHDP. It decreased the SLASHDP cost by a factor of 1.5 at most, obtained for the arm of the lowest dimensionality and decreasing to just 1.1 for the 144-dimensional arm. The opposite trend was observed for the RRT, where the higher-dimensional cases were the farthest from optimality.

The total time spent processing the path with each method, including post-processing, is given in Figure 4(c). The performance of SLASHDP was characterized by a large fixed cost of performing DP on a 100x100x10x100 lattice and the cost of computing the sampled cost values and gradients, which scales linearly in the dimension, and which dominated the cost for large dimensions. Finding an initial plan with SLASHDP was therefore quite expensive compared to the other methods for small dimensions, but less so for large dimensions. Looking at the total time including post-processing, however, yields a very different picture. Though the RRT is still faster for small dimensions, SLASHDP is roughly twice as fast for dimensions greater than 36. This is due to the fact that the elastic band spends an inordinate amount of time optimizing the RRT solution—which is very far from even local optimality—while it terminates very quickly for the SLASHDP solution, which is probably already close to optimal.



(a) Initial solution cost (b) Post-smoothing cost (c) Elapsed time

Fig. 4. Results of experiments comparing different methods applied to very high-dimensional arm planning problem pictured in Figure 3, as a function of problem dimension. 4(a) shows cost of solution produced by each method. 4(b) shows cost of each solution after post-processing with elastic band. 4(c) shows total time to find each solution, including post-processing. Note log scale of y axes. Paths produced by SLASHDP, even without post-processing, are much superior to those produced by the other methods with post-processing included. SLASHDP solutions also decrease in cost as dimension is increased. The other methods are faster for low-dimensional problems, but ours easily outperforms RRT + post-processing for very high-dimensional problems.

B. Planning for a deformable robot

We also evaluated SLASHDP on a challenging high-dimensional deformable robot problem. We assume the robot 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, we assume 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. We assume 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.

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

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

. We define a cost function $C(x)$ that penalizes the proximity of any point on the boundary to an obstacle:

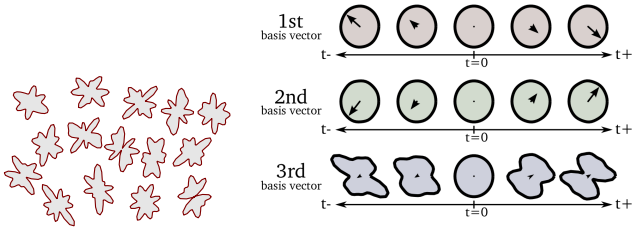
$$C(x) = 1 + e^{-(d_{obs}(x) - d_0)/\bar{d}}$$

For computational purposes, we choose K evenly-spaced $\theta_k \in [0, 2\pi)$ 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 our cost functional as

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

. We then applied SLASHDP to the problem in this form.

Figure 5 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 Figure 6, which consists of corridors running in the



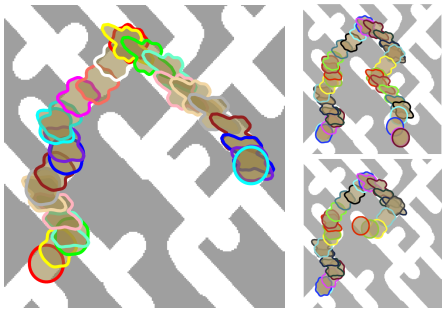
(a) Random configurations (b) Learned basis

Fig. 5. Visualization of random configurations vs. learned basis for deformable robot planning problem. Randomly sampled configurations are shown for reference in 5(a). Learned basis vectors are displayed in 5(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.

"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. This is confirmed in the generated paths, in which the robot deforms between these shapes, transitioning through a circular phase at the corners.

To demonstrate the usefulness of SLASHDP to generate a feedback plan, we initially generated a path using a query pair of configurations, depicted in Figure 6(a). We then used the generated value function to quickly produce solutions for a variety of final configurations, two of which are shown in Figure 6(b). Although the initial plan was very costly to create (on the order of tens of minutes), subsequent plans were generated very quickly (on the order of milliseconds) thanks to the availability of the value function.

Quantitative results are given in Figure 7. For this series of experiments, we tested the ability of SLASHDP both to find an initial path and to post-process a path by the *generalized shortcuts* heuristic presented in Section III-D. As a baseline, we computed a plan for the center of mass that merely translated the robot through the maze without deformation. We then attempted to post-process this path with different methods in order to lower its cost. Neither an elastic band



(a) Queried plan (b) Extra plans

Fig. 6. Visualization of results of experiment in planning for deformable robot. Figure 6(a) shows plan obtained by planning for a given query configuration. Fig. 6(b) shows additional plans obtained very quickly from the value function computed for 6(a).

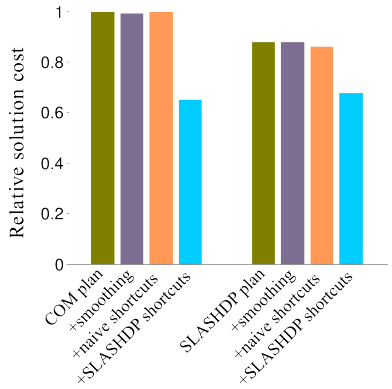


Fig. 7. Results of planning for a deformable robot. Left group of bars represent cost of plans obtained by using different methods to post-process an initial plan generated for the robot's centroid with no deformation (bar labeled *COM plan*). Right group of bars represent cost of plans obtained by post-processing initial plan generated by SLASHDP (bar labeled *SLASHDP*). *+smoothing* indicates post-processing with an elastic band, *+naive shortcuts* post-processing with a straight-line shortcut heuristic, and *+SLASHDP shortcuts* post-processing by using SLASHDP to generate local shortcuts.

nor the naive shortuct heuristic described in Section III-D was able to significantly decrease the cost of this initial plan. Post-processing with SLASHDP shortcuts, however, reduced the cost of this plan by nearly 40%.

Similar results were obtained when we started with an initial plan generated by SLASHDP. The initial path was a modest 10% better than the path obtained via translation-only planning. This path was not significantly improved by post-processing with an elastic band or naive shortcuts. SLASHDP shortcuts, however, again yielded a large decrease in the objective value, settling to a value very close to that obtained by smoothing the purely translational path with SLASHDP.

V. Conclusions

We have presented SLASHDP, a novel learning-based approach to solving high-dimensional holonomic motion planning problems using dynamic programming. SLASHDP efficiently produces high-dimensional motion plans that are of a quality far surpassing what can be obtained with traditional global search + smoothing methods. Furthermore, by the nature of its use of dynamic programming, we additionally obtain a value function that can be used to generate a feedback optimal control law. We have also demonstrated how SLASHDP can be used as a powerful generalization of the naive shortcut heuristic typically used for path post-processing. Experimental results have shown the method's ability to scale effortlessly to very large-dimensional arm planning problems while producing very high-quality solutions. Results from a deformable robot planning problem also show the method's ability to serve as an exceptional post-processing method by exploiting local low-dimensional structure.

In the near future, we would like to explore how best to extend SLASHDP to handle nonholonomic constraints. We are currently investigating the use of the SLASHDP value function as an oracle or heuristic in the context of heuristic search methods such as A^*, which can leverage this heuristic while taking nonholonomic constraints into account. Similarly, we anticipate that the SLASHDP value might be usable as a distance function for randomized planning methods.

Our hope is that this work will stir interest in the systematic discovery of structure in planning problems of different sorts, which we anticipate is a direction that will produce interesting advances in the future.

References

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

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

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

[4] N. M. Amato and Y. Wu, "A randomized roadmap method for path and manipulation planning," in *Proceedings IEEE International Conference on Robotics & Automation*, 1996, pp. 113–120.

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

[6] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," *International Journal Computational Geometry & Applications*, vol. 4, pp. 495–512, 1999.

[7] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *NIPS*, 2004.

[8] M. Likhachev and A. Stentz, "R* search," in *AAAI*, 2008.

[9] R. E. Korf, "Real-time heuristic search," *Artificial Intelligence*, vol. 42, no. 2-3, pp. 189 – 211, 1990.

[10] A. E. Prieditis, "Machine discovery of effective admissible heuristics," *Machine Learning*, 1993.

[11] N. D. Ratliff, "Learning to search: structured prediction techniques for imitation learning," Ph.D. dissertation, Carnegie Mellon Univeristy, 2009.

[12] E. Plaku and L. E. Kavraki, "Quantitative analysis of nearest neighbors search in high-dimensional sampling-based motion planning," in *WAFR*, New York, NY, 2006.

[13] I. A. Sucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *WAFR*, Guanajuato, Mexico, 2008.

[14] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," *PNAS*, vol. 93, no. 4, pp. 1591–1595, 1996.

[15] R. Geraerts and M. Overmars, "Clearance based path optimization for motion planning," in *ICRA*, 2004, pp. 2386–2392.

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