2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

# Predicting Initialization Effectiveness for Trajectory Optimization

Jia Pan Zhuo Chen Pieter Abbeel

***Abstract*— Trajectory optimization is a method for solving motion planning problems by formulating them as non-convex constrained optimization problems. The optimization process, however, can get stuck in local optima that are in collision. As a consequence, these methods typically require multiple initializations. This poses the problem of deciding which initializations to use when given a limited computational budget. In this paper we propose a machine learning approach to predict whether a collision-free solution will be found from a given initialization. We present a set of trajectory features that encode the obstacle distribution locally around a robot. These features are designed for generalization across different tasks. Our experiments on various planning benchmarks demonstrate the performance of our approach.**

## I. Introduction

Trajectory optimization algorithms are becoming attractive options for robotic motion planning, especially for problems with many degrees of freedom (DOF). Given an initial trajectory that may contain collisions and violate constraints, trajectory optimization methods can often quickly converge to a high-quality, locally-optimal solution. These methods are readily able to incorporate dynamics, smoothness and obstacle avoidance.

Despite their success in various applications, trajectory optimization techniques suffer from a critical limitation: their performance heavily depends on the choice of initial trajectories. Certain initializations passing through obstacles in unfavorable ways may get stuck in infeasible solutions and cannot resolve all collisions in the final outcome, as illustrated in Figure 1. For instance, trajectories that pass through the medial axis of an obstacle are prone to getting stuck in a local optimum.

We propose a learning-based approach to predict the quality of a trajectory as an initialization to the optimization-based motion planning. Here, a trajectory's quality is measured by whether the optimization problem converges to a collision-free trajectory when using the given trajectory as the initial guess. In other words, the quality is defined in a qualitative manner, i.e. 'good' or 'bad'. For the learning algorithm, we first design a set of trajectory features, which are relevant to a trajectory's effectiveness as an initialization. These features are both task-independent and scene-independent, and hence they are transferable among different planning tasks. We use four types of features: (i) spatial signed distance (SSD) vectors for each robot link; (ii) difference between SSD vectors for the same robot link in adjacent trajectory waypoints and for adjacent robot links

Authors are with the Department of Electrical Engineering and Computer Science, University of California at Berkeley, CA, USA. {jia.pan, zhuo, pabbeel}@berkeley.edu

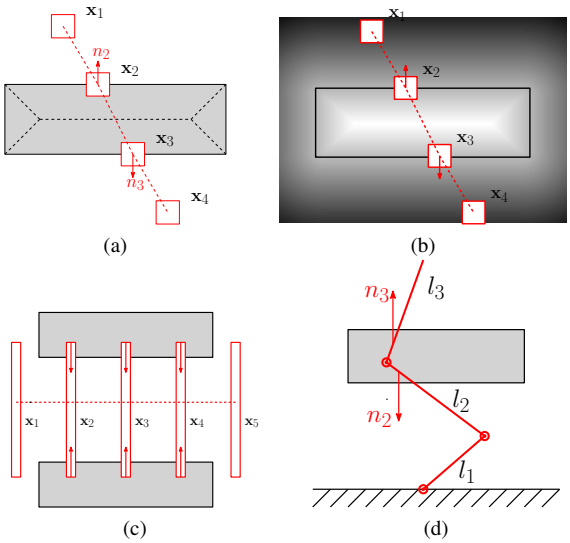



Fig. 1. Illustration of typical reasons for trajectory optimization to get stuck in local optima that are not collision-free. The gradient based on penetration depth (a) or distance fields (b) may push waypoints in in-consistent directions. When a robot collides simultaneously with multiple obstacles (c), the robot may get stuck in an infeasible local optimum since different obstacles push the robot in different directions. For a robot with multiple links (d), the gradient may result in in-consistent directions for different links. $\mathbf{x}_i$ in these figures denote different trajectory waypoints.

in the same waypoint; (iii) spherical harmonics transform of SSD vectors; and (iv) features measuring the non-convexity of the constrained optimization problem in the trajectory neighborhood. These features try to capture the obstacle distribution locally around a robot trajectory and thus can be expected to be useful for predicting a trajectory's behavior for a trajectory optimization algorithm. We calculate these features for a set of randomly selected trajectories for different tasks in various benchmarks. We then run optimization-based motion planning using these trajectories as initial guesses and obtain a labeled dataset. In the dataset, each trajectory is augmented with a 'good' or 'bad' label according to whether the optimization converges to a collision-free trajectory. With this dataset as the training set, we learn a prediction function for trajectory effectiveness. We evaluate the accuracy of the learned prediction function by counting the ratio of correct predictions for a set of new trajectories on various benchmarks.

## II. Related Work

Trajectory optimization is an established method in robotics to generate a high quality path from an initial trajectory that may be in-collision or dynamically infeasible. It has also been used as a post-processing phase to remove redun-

dant or jerky motion in trajectories generated by traditional planning algorithms [1], [2]. Trajectory optimization works by solving an optimization problem on trajectory space. Many approaches use a spline-based or waypoint-based representation for trajectory and use cost gradient information for minimization, including CHOMP [3], STOMP [4] and TrajOpt [5]. There is extensive, closely-related work in optimal control, which focuses less on collisions and more on systems with complicated dynamical properties. Some of the most notable work includes differential dynamic programming [6], [7], iterative LQR [8], approximate inference control [9] and optimization-based control involving contacts [10], [11], [12], [13].

One main challenge of trajectory optimization is its sensitivity to the choice of initial guesses and may get stuck in infeasible local optima, since the collision-free constraint is highly non-convex.

For general non-convex optimization problems, the dependence on initialization is also a well-known challenge. A simple and popular solution is using multiple random initializations to help the algorithms escape from bad local optima. In recent work [14] machine learning was used to build the relationship between the starting point of an optimization algorithm and the objective value of the final outcome. The main difference is that our approach is for trajectory optimization, where generating a feasible trajectory is more important than decreasing the absolute cost of the objective function. Moreover, trajectory optimization can exploit workspace heuristics for selecting good initial guesses, and such heuristics are not available in general optimization problems. Another closely related line of work is about trajectory prediction [15], where learning algorithms are used to predict a good path in a new environment from a database of demonstrated trajectories. The predicted result may not be feasible and hence requires a second step to resolve all violations.

Prior work has considered designing trajectory features for various applications. For example, Bentivegna et al. [16] and Stolle et al. [17] represented a library of trajectories by feature vectors for control policy transfer. Berenson et al. [18] designed several criteria and features to evaluate whether a trajectory from a path library is likely to be reused in a new planning scene after suitable repair operations. Trajectory features are also used to predict class labels of moving objects based on their trajectories [19]. Moreover, features have been designed to compress trajectory libraries [20]. However, all these features only capture properties of a trajectory itself. In contrast, the features proposed in our paper encode the interaction between the trajectory and the surrounding environment, which is critical for predicting how fast a trajectory can be pushed away from the obstacles by a trajectory optimization algorithm.

## III. Problem Definition

### A. Background

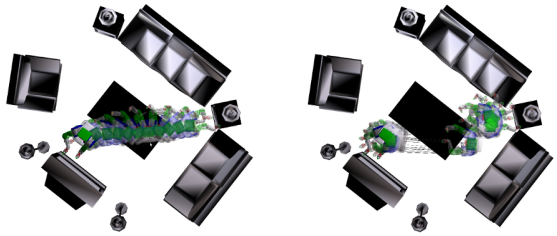
Robotic motion planning problems can be formulated as non-convex optimization problems, i.e., minimize an objective subject to inequality and equality constraints:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{x}) \leq 0,\ i = 1, 2, \ldots, n_{\text{ieq}} \\ & h_i(\mathbf{x}) = 0,\ i = 1, 2, \ldots, n_{\text{eq}} \end{aligned} \tag{1}$$

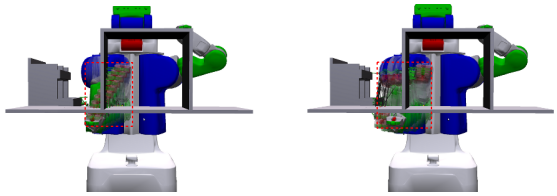
where $f$, $g_i$ and $h_i$ are scalar functions. For planning problems that involve only kinematics and represent the trajectory as a sequence of $T$ waypoints, the optimization variable $\mathbf{x}$ is of the form $\mathbf{x} = \boldsymbol{\theta}_{1:T}$, where $\boldsymbol{\theta}_t \in \mathbb{R}^K$ denotes the configuration at the $t$-th waypoint for a system with $K$ degrees of freedom. For problems with dynamics, the optimization variable $\mathbf{x}$ may also include velocities $\dot{\boldsymbol{\theta}}_t$ and torques $\boldsymbol{\tau}_t$.

Collision avoidance is one of the most important inequality constraints in motion planning. But it is difficult to be formulated in a closed form and hence is challenging for optimization. Various approximations for collision avoidance constraints have been proposed, including distance fields [21], [3], penetration depth [22], and swept volume [5]. Besides collision avoidance, common inequality constraints include kinematic constraints (e.g., joint limits), kinodynamic constraints (e.g., bounded velocity or acceleration) or dynamic constraints (e.g., dynamic stability constraints). One example of equality constraints is the end-effector pose constraint where the robot must reach a target pose.

The objective function $f(\mathbf{x})$ is often chosen to be a quadratic form of $\mathbf{x}$. For example, a widely used objective function for kinematic planning is $f(\boldsymbol{\theta}_{1:T}) = \sum_{t=1}^{T} \|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|^2$, which encourages minimum-length or smoothness in the final outcome [5], [3].



(a) initial path (b) result path



(c) initial path (d) result path

Fig. 2. Failure cases when using the state-of-the-art trajectory optimization method [5] for motion planning. (a) shows the initial path for a whole-body planning, which passes through the medial axis of the desk obstacle. (b) is the trajectory optimization outcome, which is stuck in an infeasible condition and the trajectory is pulled apart by virtue of being on both sides of an obstacle. (c) shows the initial path for the arm planning and the collision cannot be resolved in the final trajectory shown in (d). The main difference between trajectories in (c) and (d) is marked by dashed red boxes.

Trajectory optimization is a challenging non-convex problem, and many approaches have been presented to solve

it effectively. Among them are sequential convex optimization [5], covariant Hamiltonian optimization [3], and stochastic optimization [4]. However, given an initial trajectory that contains collisions, none of these methods are guaranteed to find a collision-free solution due to the non-convex constraints in the optimization. Figure 1 shows some scenarios illustrating how trajectory optimization tends to get stuck in local optima that are not collision-free, and Figure 2 provides failure cases when using the state-of-the-art trajedctory optimization method [5] in these scenarios.

It is an on-going line of research to reshape the non-convex optimization formulations to improve their performance. In this paper we consider the following complimentary problem: Given a trajectory optimization approach, how to predict whether an initialization will result in a collision-free solution or not. Such predictive capabilities would enable selecting a good initial trajectory for motion planning so that trajectory optimization is likely to converge to a collision-free local optimum. Such information can also be used to improve the performance of trajectory optimization being run in parallel for multiple initializations: For each solution sequence starting with a different initial guess, we can stop bad solution sequences early enough and focus computational budgets on solution sequences that potentially can converge to good outcomes.

### B. Outline of Proposed Approach

The problem to predict whether a given trajectory is a good or bad initial guess for a trajectory optimization algorithm can be formalized as a two-class classification problem. In fact, a trajectory is good if and only if it locates in the basin of attraction of a local optimum that is collision-free.

For the classification problem, we first design trajectory features that are potentially useful to distinguish good and bad initializations. The features extracted from a trajectory $\boldsymbol{\theta}_{1:T}$ are denoted as a vector $\mathbf{z} = E(\boldsymbol{\theta}_{1:T})$, where $E(\cdot)$ is the extraction function. For each trajectory, we use a binary variable $y$ to denote whether the trajectory optimization succeeds ($y = 1$, a good initial guess) or fails ($y = 0$, a bad initial guess).

Given a set of $N$ trajectories, we build a training set $\{\mathbf{z}_i, y_i\}_{i=1}^N$. Based on the data set, we learn a classifier $C(\mathbf{z})$, which is the predictor for a trajectory's effectiveness. Given a new trajectory as input, $C(\cdot)$ will provide its qualitative evaluation (good or bad) as output. The learned predictor $C(\cdot)$ can depend on the planning methods, because different optimization approaches may differ in for which initial trajectories they end up finding feasible solutions. However, our learning method is general and does not depend on the type of trajectory optimization method.

## IV. Predicting Initialization Effectiveness for Trajectory Optimization

In this section, we design trajectory features to distinguish good and bad initializations and use the extracted features for predicting the effectiveness of a new trajectory.

Before extracting features from a trajectory, we first perform re-sampling on it so that waypoints are uniformly distributed between two end-points. The re-sampling process removes some the parameterization-dependent differences between trajectories and hence makes it easier to compare them. After re-sampling, each trajectory has $T$ waypoints. This also results in fixed-length feature vectors.

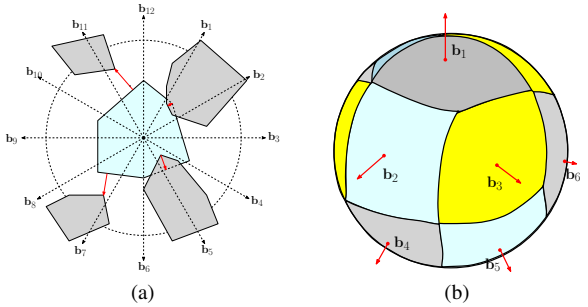
### A. Background: Signed Distances

We design features to encode the obstacle distribution locally around a trajectory, as such information is closely related with a trajectory's effectiveness. Intuitively speaking, if a trajectory is deep inside the obstacles, it is less likely to be pushed out of them; whereas if it is mostly outside the obstacles, it should converge to a collision-free solution. To quantify the obstacle distribution surrounding a trajectory, we use signed distances and normals, which are used in previous trajectory optimization algorithms such as [5]. Informally, the signed distance between two objects is the length of the smallest translation that puts them in contact; the signed normal is the direction of such translation. The signed distance can be efficiently computed for convex objects using GJK algorithm [23] and EPA algorithm [24]. For use with non-convex objects, these objects can be represented as a set of convex objects together making up the original non-convex object.

### B. Feature I: Spatial Signed Distance Vectors

Based on the signed distances formulation, a natural choice for designing trajectory feature vector is to collect signed distances and normals between all pairs of robot links and environment obstacles, for each waypoint in the trajectory. More formally, given a robot with $n$ links $\mathcal{A}_i$ in an environment with $m$ obstacles $\mathcal{B}_j$, the feature vector would be $\mathbf{z}_{\text{sd}} = \{\text{sd}_{i,j,t}, \mathbf{n}_{i,j,t}\}_{1\le i\le n, 1\le j\le m, 1\le t\le T}$, where $\text{sd}_{i,j,t} = \text{sd}(\mathcal{A}_i(t), \mathcal{B}_j)$ is the signed distance value between $\mathcal{A}_i$ and $\mathcal{B}_j$ at waypoint $\boldsymbol{\theta}_t$, and $\mathbf{n}_{i,j,t}$ is the corresponding normal. This feature vector is of dimension $4 \times m \times n \times T$ and provides a complete description for the obstacle distribution around the trajectory. However, the dimension of this feature depends on $m$, the number of obstacles in the environment, which makes this feature not transferable among different environments. Moreover, in many real-world systems, there may be a large number of obstacles (e.g., the obstacles from sensor data are usually represented as thousands of boxes [25]). This would result in a long feature vector and brings challenges for both data storage and the following learning procedure. Thus we propose to summarize this information in a more compact manner that lends itself better to generalization.

Our solution is to re-organize the signed distances and normals into a new form called the Spatial Signed Distance (SSD) vectors. We illustrate the basic idea of SSD via a 2D example in Figure 3(a). We first partition the space around a robot link using a dictionary of direction vectors $D = \{\mathbf{b}_1, ..., \mathbf{b}_M\}$. For the 3D case, we generate the direction dictionary using the approach introduced in [26], which computes a uniform deterministic sequence of samples over




(a) (b)

Fig. 3. Spatial signed distance vectors: (a) shows a 2D example about SSD vector generation. The surrounding space around a robot link (the cyan color part) is uniformly divided by a spatial vector dictionary with 12 directions $\{\mathbf{b}_1, ..., \mathbf{b}_{12}\}$. The signed distance value $\text{sd}_j$ between the robot and the $j$-th obstacle (the grey part) is then accumulated toward a direction $\mathbf{b}_k$, which has the smallest angle with the corresponding signed distance normal $\mathbf{n}_j$, i.e., with the smallest $|\mathbf{n}_j \cdot \mathbf{b}_k|$. (b) In 3D case, we use the approach introduced in [26] to generate dictionary vectors that make equisurface partition on a sphere surface.

$S^2$ (as shown in Figure 3(b)) using a specific multi-resolution grid structure. This partitions the sphere surface into bins of the same size.

Given the signed distances and normals between the robot link and all the obstacles in the environment, we accumulate the signed distances along these discretized prototype directions in dictionary $D$ and the result is a $M$-dimensional vector $\mathbf{z}_{\text{ssd}}$. The $k$-th element of $\mathbf{z}_{\text{ssd}}$ is denoted as $\mathbf{z}_{\text{ssd}}[k]$, which summarizes signed distances along direction $\mathbf{b}_k$. If $\mathbf{z}_{\text{ssd}}[k]$ is positive, then the link still has some distance from the obstacle and is safe; if $\mathbf{z}_{\text{ssd}}[k]$ is negative, then the link will go deeper inside the obstacle when being pushed in this direction and hence should be pushed in other directions.

The algorithm to compute $\mathbf{z}_{\text{ssd}}$ based on signed distance information is shown in Figure 4. First, each element in $\mathbf{z}_{\text{ssd}}$ is initialized by a default value $R > 0$, which defines the size of the local neighborhood around the link. We set $R$ to be the robot link's bounding radius; this implies that a large object would have a larger local neighborhood. Next, suppose the $j$-th obstacle's signed distance to the robot link is $\text{sd}_j$ and the corresponding normal is $\mathbf{n}_j$. We first find the dictionary direction $\mathbf{b}_k$ which has the smallest angle with $\mathbf{n}_j$, i.e., quantizing $\mathbf{n}_j$ to the prototype direction $\mathbf{b}_k$ with the smallest $|\mathbf{n}_j \cdot \mathbf{b}_k|$. Next, we merge $\text{sd}_j$ with the existing value in $\mathbf{z}_{\text{ssd}}[k]$. If both $\text{sd}_j$ and $\mathbf{z}_{\text{ssd}}[k]$ are negative, i.e., a new penetration is detected in a direction where the penetration was detected before, we add $\text{sd}_j$ onto $\mathbf{z}_{\text{ssd}}[k]$ to give a large penalty for such in-collision case. For other cases, we update $\mathbf{z}_{\text{ssd}}[k]$ by $\text{sd}_j$ if $\mathbf{z}_{\text{ssd}}[k]$'s current value is larger than $\text{sd}_j$. In other words, $\mathbf{z}_{\text{ssd}}[k]$ is with the signed distance value of the closest obstacle. If $\mathbf{z}_{\text{ssd}}[k]$ remains $R$ after the computation, this means that no penetration is detected within the given link's neighborhood along direction $\mathbf{b}_k$. In this way, we summarize the obstacle distribution information around a robot link into a length-$M$ SSD vector.

After computing SSD vectors for all links belonging to all trajectory waypoints, we concatenate them together into the SSD feature vector for the entire trajectory, which is

**Input**: Signed distance values $\text{sd}_j$ and normals $\mathbf{n}_j$ between the robot link and the $m$ obstacles in the environment; The spatial vector dictionary of size $M$: $D = \{\mathbf{b}_1, ..., \mathbf{b}_M\}$
**Output**: Spatial signed distance vector $\mathbf{z}_{\text{ssd}}$

Initialize $\mathbf{z}_{\text{ssd}}$ as a dimension-$M$ vector with default value $R$;
**for** $j \leftarrow 1$ to $m$ **do**
  Find $\mathbf{b}_k$ in $D$ where $|\mathbf{b}_k \cdot \mathbf{n}_j|$ is the smallest;
  **if** $\text{sd}_j < 0$ and $\mathbf{z}_{\text{ssd}}[k] < 0$ **then**
    $\mathbf{z}_{\text{ssd}}[k] \leftarrow \mathbf{z}_{\text{ssd}}[k] + \text{sd}_j$;
  **else**
    $\mathbf{z}_{\text{ssd}}[k] \leftarrow \min(\mathbf{z}_{\text{ssd}}[k], \text{sd}_j)$;
  **end if**
**end for**

Fig. 4. SSD vector generation for one robot link at a trajectory waypoint

of dimension $n \times M \times T$ and is also denoted as $\mathbf{z}_{\text{ssd}}$. For environments with many obstacles, $M \ll m$ and therefore $\mathbf{z}_{\text{ssd}}$ is more compact than $\mathbf{z}_{\text{sd}}$.

### C. Feature II: Difference between Spatial Signed Distance Vectors

The spatial signed distance vector provides a description of the obstacle distribution around the trajectory on a per link basis. However, as we mentioned in Section III, whether the signed distance normal is consistent between adjacent links or adjacent waypoints in an initial trajectory is also important for the convergence of a trajectory optimization algorithm. Thus, we include features to measure the consistency between two spatial signed distance feature vectors. Given two vectors $\mathbf{z}_{\text{ssd}}{}^a$ and $\mathbf{z}_{\text{ssd}}{}^b$, we evaluate their consistency using their dot-product $\mathbf{z}_{\text{ssd}}{}^a \cdot \mathbf{z}_{\text{ssd}}{}^b$. This dot-product has a larger value when $\mathbf{z}_{\text{ssd}}{}^a[k]$ and $\mathbf{z}_{\text{ssd}}{}^b[k]$ are of the same sign for all $k$. And it has a smaller value when some $\mathbf{z}_{\text{ssd}}{}^a[k]$ and $\mathbf{z}_{\text{ssd}}{}^b[k]$ are of the opposite sign, i.e., the signed distance normals are not consistent in direction $\mathbf{b}_k$. We compute dot-product between SSD vectors of the same link in adjacent waypoints and between adjacent links in the same waypoint. We collect all these results relating to consistency in the signed distances in a feature vector denoted by $\mathbf{z}_{\text{con}}$.

### D. Feature III: Spherical Harmonics Spatial Feature

The spatial signed distance feature is not rotation-invariant. As a result when, for example, both the environment and the trajectory are rotated by a given angle, the feature vector will be completely different. This could affect its generalization capability. Another example is when the environment is symmetric (such as Figure 6(c)). Overall, the lack of rotation-invariance in the feature may decrease the prediction accuracy of the learning algorithm.

As shown in Figure 3(a), the spatial signed distance feature is a function defined on a sphere around a robot link. To represent the feature in a rotation-invariant manner, we utilize spherical harmonics. The same idea has been used in various applications, including computer vision [27] and graphics [28].

The theory of spherical harmonics says that any spherical function $s(\theta, \phi)$ can be expressed as a sum of complex spherical harmonic basis functions $Y_l^m$:

$$s(\theta, \phi) = \sum_{l=1}^{\infty} \sum_{m=-l}^{l} a_l^m Y_l^m(\theta, \phi). \quad (2)$$

The amplitudes of the harmonic coefficients $\|a_l^m\|$ are invariant to any rotation in the azimuthal direction. Moreover, the amplitude of $s_l$, the $l$-th frequency component of the function $s$, is rotation-invariant under any rotations, where $s_l = \sum_{m=-l}^{l} a_l^m Y_l^m(\theta, \phi)$. The magnitude of $s_l$ can also be computed based on $a_l^m$: $\|s_l\|^2 = \sum_{m=-l}^{l} \|a_l^m\|^2$. We use $\|s_l\|$ as features in 3D benchmarks. In 2D case, we use $\|a_l^m\|$ as features since only azimuthal rotation is allowed in 2D and $\|a_l^m\|$ can provide richer information. Similar to previous work [27], [28], we choose a bandwidth $b$ and store only $b$ lowest-frequency components in our spherical harmonics feature $\mathbf{z}_{\text{sht}}$. More formally, the feature is defined as $\mathbf{z}_{\text{sht}} = \{\|s_l\|\}$ in 3D and $\mathbf{z}_{\text{sht}} = \{\|a_l^m\|\}$ in 2D, where $m = 0, ..., l$ and $l = 0, ..., b$.

### E. Feature IV: Convexity Features

As another category of features, we also measure the local convexity of the trajectory optimization problem around the initial trajectory. The optimization problem formalized in Equation 1 usually involves a non-convex objective and non-convex constraints. In trajectory optimization algorithms, these non-convex terms are approximated by convex functions in each iteration of the optimization. The convexity features quantify how close the approximations are in such 'convexify' steps.

We first compute Hessian matrices of the objective and constraint functions at the trajectory. Next, to measure the local non-convexity of the optimization problem, we compute the following features based on the eigenvalues of the Hessian matrices: (i) the minimum eigenvalue; (ii) the sum of negative eigenvalues; (iii) the maximum eigenvalue; (iv) the sum of all eigenvalues. We compute these four statistics for each objective and constraint function, and generate a total of $4 \times T \times (1 + n_{\text{ieq}} + n_{\text{eq}})$ convexity features, which are collected in a feature vector denoted as $\mathbf{z}_{\text{conv}}$.

### F. Trajectory Evaluation and Effectiveness Prediction

To evaluate whether a trajectory is 'good' or 'bad' as the initial guess for the optimization, currently we use two simple criteria: a trajectory is justified as a 'good' initialization if the optimization algorithm converges and the final outcome does not violate any constraints.

Given a set of trajectories with their features and evaluation results, we can use any two-class classification algorithm to learn a classifier for the trajectory. As the dimension of our trajectory feature is high (larger than 200), we choose the linear support vector machine [29] as the classifier due to its simplicity and efficiency.

## V. Experiment and Evaluation

We experimentally evaluate the trajectory features that we designed and demonstrate the accuracy of our effectiveness prediction approach on various 2D and 3D benchmarks, as shown in Figure 6, 7 and 8. The trajectory optimization algorithm we used in the experiment is TrajOpt [5], which is one of the state-of-the-art trajectory optimization algorithms.

### A. Experimental Setting

For each benchmark, we have a pre-defined set of task settings (the initial and goal configurations) as seed tasks. For the PR2 benchmarks, we use the 198 arm planning problems and 96 full-body problems used in TrajOpt [5]. For the Dubins car benchmarks, we manually select a set of initial-goal configurations that should have feasible solutions. Next, we perturb the initial and goal configurations around these seed tasks to generate more random tasks. We filter out the invalid tasks where the initial or goal configurations are in-collision. We also remove the trivial cases where the linear interpolation between initial and goal is collision-free. We did not randomly select tasks in the entire configuration space because this will incorrectly bias the classifier toward tasks that will hardly happen in real world robotics applications.

Given one planning task, we first generate a trajectory which is a simple linear interpolation between the initial and goal configurations. Next, we perform random perturbation of the trajectory waypoints and generate more random trajectories. For each of these trajectories, we run the trajectory optimization algorithm using it as the initial guess. When the optimization stops, we will obtain a sequence of trajectories. These intermediate trajectories can also be used as initializer and they would all have the same effectiveness label as the actual initial guess. As a result, if the optimization converges and the final outcome is feasible, we evaluate the given initial trajectory and all intermediate trajectories as 'good' initializations. Otherwise we evaluate all of them as 'bad'. Finally, we extract features for all these labeled trajectories and add them into the dataset. In this way, we generate more than $10,000$ trajectories for each planning benchmark. For each benchmark, we use half of the data for training a classifier and the rest for testing experiments. When splitting training and test sets, we make sure that trajectories corresponding to the same task are assigned into the same set, in order to guarantee that the test set is independent of the training set. In Figure 5, we visualize the features of two trajectories from different benchmarks. Except $\mathbf{z}_{\text{conv}}$, all these features can be computed in less than 10 milliseconds per trajectory. $\mathbf{z}_{\text{conv}}$ is more expensive (about 8 seconds) since it is time consuming to compute a trajectory's Hessian matrix numerically.

### B. Effectiveness Prediction on Same Benchmarks

We first show the prediction accuracy when using a classifier learned on one benchmark to predict the effectiveness of trajectories in the same benchmark but for different tasks. The results are shown in Table III, where we compare the

baseline accuracy and the accuracy when different combinations of features are used to train a classifier. From the results, we observe that the learned classifiers provide accuracy significantly higher than the baseline (which simply predicts according to the majority label in the training set). However, the peak accuracy may be achieved by a different combination of features on different benchmarks.

### C. Effectiveness Prediction on Different Benchmarks

A more challenging test is to check whether a classifier learned on one benchmark is able to correctly predict effectiveness for trajectories from other benchmarks. For this test, we only obtain partial success: We indeed observe several successful transfers among different benchmarks as shown in Table IV. However, in most cases, the prediction accuracy is low when transferring between different benchmarks.

There are several reasons for the poor transfer performance of the current prediction algorithm. First, our current feature design only considers the spatial information locally around the trajectory and does not take into account the global description about the environment, which has been proved to be helpful for trajectory transferring between different environments [15]. For instance, the classifier learned on squared-dubin in Figure 6(c) can be successfully transferred to jagged-dubin in Figure 6(a) because the environments are similar. The transfer to gap-dubin in Figure 6(b) is difficult because the existence of the narrow gap, which does not exist in the squared-dubin or jagged-dubin benchmark. Second, a high-DOF robot may use different part of DOFs in different scenarios; this results in different trajectory distributions on different benchmarks. Hence, we need more trajectory data from various scenarios and tasks, in order to completely cover the trajectory space. Finally, the rotation-invariance of features is also important for transferring. From the successful case shown in Table IV, we observe that the feature combination $\mathbf{z}_{\text{con}} + \mathbf{z}_{\text{sht}}$ behaves the best in transferring, and both of them are rotation-invariant features.

## VI. Conclusion

In this paper, we have introduced novel features for trajectory effectiveness and described a learning-based approach to predict whether an initialization will lead to a collision free path. We evaluated the accuracy of our approach on different planning tasks within various benchmarks. Additionally, we showed initial results on transferring the learned classifier to other benchmarks.

This work serves as a baseline of initial trajectory selection for trajectory optimization based motion planning. Future work could consider more combinations of trajectory features and explore more criteria for trajectory evaluation, including running time, objective value and cost for violated constraints. Moreover, instead of the qualitative metric used in this paper, it would be of interest to apply regression algorithms to compute a quantitative evaluation for trajectory's effectiveness, which can be used for trajectory ranking during the optimization. Finally, it would be of interest to integrate our approach with trajectory optimization.

## Acknowledgments

This research has been funded in part by Darpa Young Faculty Award #D13AP00046 and by a Sloan Fellowship.

## References

[1] J.-C. Latombe, *Robot Motion Planning*. Norwell, MA, USA: Kluwer Academic Publishers, 1991.

[2] S. M. LaValle, *Planning Algorithms*. New York, NY, USA: Cambridge University Press, 2006.

[3] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: gradient optimization techniques for efficient motion planning," in *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, 2009, pp. 4030–4035.

[4] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Proceedings of International Conference on Robotics and Automation*, 2011, pp. 4569–4574.

[5] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization," in *Proceedings of Robotics: Science and Systems*, 2013.

[6] D. Jacobson and D. Mayne, *Differential Dynamic Programming*. Amsterdam: Elsevier, 1970.

[7] C. G. Atkeson, "Using local trajectory optimizers to speed up global optimization in dynamic programming," in *Proceedings of Advances in Neural Information Processing Systems*, 1994, pp. 663–670.

[8] E. Todorov and W. Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of fAmerican Control Conference*, 2005, pp. 300–306 vol. 1.

[9] M. Toussaint, "Robot trajectory optimization using approximate inference," in *Proceedings of International Conference on Machine Learning*, 2009, pp. 1049–1056.

[10] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 43:1–43:8, July 2012.

[11] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Proceedings of International Conference on Intelligent Robots and Systems*, 2012, pp. 4906–4913.

[12] T. Erez and E. Todorov, "Trajectory optimization for domains with contacts using inverse dynamics," in *Proceedings of International Conference on Intelligent Robots and Systems*, 2012, pp. 4914–4919.

[13] M. Posa and R. Tedrake, "Direct trajectory optimization of rigid body dynamical systems through contact," in *Algorithmic Foundations of Robotics X, Springer Tracts in Advanced Robotics, Volume 86*, E. Frazzoli, T. Lozano-Perez, N. Roy, and D. Rus, Eds. Springer, 2013, pp. 527–542.

[14] A. Cassioli, D. Di Lorenzo, M. Locatelli, F. Schoen, and M. Sciandrone, "Machine learning for global optimization," *Comput. Optim. Appl.*, vol. 51, no. 1, pp. 279–303, Jan. 2012.

[15] N. Jetchev and M. Toussaint, "Fast motion planning from experience: trajectory prediction for speeding up movement generation," *Autonomous Robots*, vol. 34, no. 1-2, pp. 111–127, Jan. 2013.

[16] D. Bentivegna, C. Atkeson, and G. Cheng, "Learning similar tasks from observation and practice," in *Proceedings of International Conference on Intelligent Robots and Systems*, 2006, pp. 2677–2683.

[17] M. Stolle, H. Tappeiner, J. Chestnutt, and C. Atkeson, "Transfer of policies based on trajectory libraries," in *Proceedings of International Conference on Intelligent Robots and Systems*, 2007, pp. 2981–2986.

[18] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *Proceedings of International Conference on Robotics and Automation*, 2012, pp. 3671–3678.

[19] J.-G. Lee, J. Han, X. Li, and H. Gonzalez, "TraClass: trajectory classification using hierarchical region-based and trajectory-based clustering," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 1081–1094, Aug. 2008.

[20] O. Arikan, "Compression of motion capture databases," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 890–897, 2006.

[21] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings of International Conference on Robotics and Automation*, vol. 2, 1985, pp. 500–505.

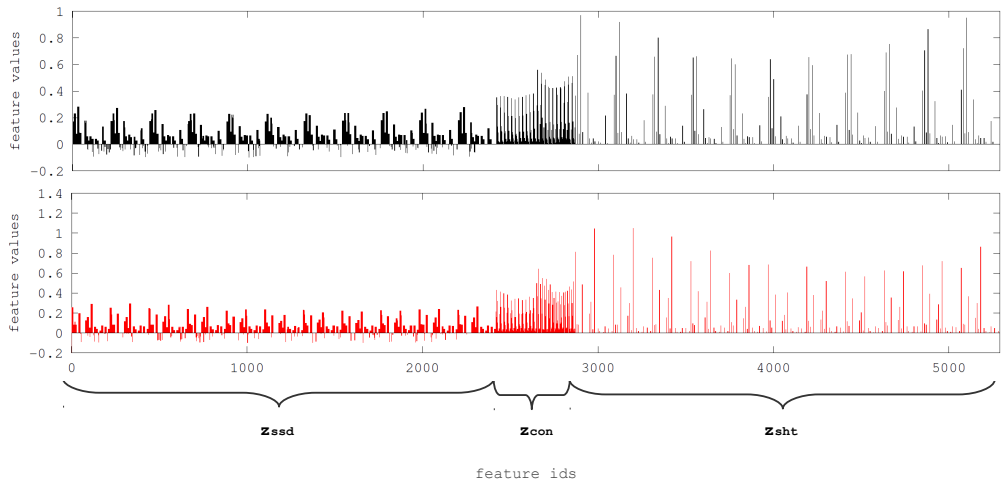



Fig. 5. $\mathbf{z}_{\text{ssd}}$, $\mathbf{z}_{\text{con}}$ and $\mathbf{z}_{\text{sht}}$ features for two different trajectories in the bookshelf benchmark. Rough periods appears for all the three sub-features, because these features are computed for each trajectory waypoint.

| | baseline | $\mathbf{z}_{\text{conv}}+\mathbf{z}_{\text{ssd}}+\mathbf{z}_{\text{con}}+\mathbf{z}_{\text{sht}}$ | $\mathbf{z}_{\text{ssd}}+\mathbf{z}_{\text{con}}+\mathbf{z}_{\text{sht}}$ | $\mathbf{z}_{\text{ssd}}+\mathbf{z}_{\text{con}}$ | $\mathbf{z}_{\text{con}}+\mathbf{z}_{\text{sht}}$ | $\mathbf{z}_{\text{ssd}}+\mathbf{z}_{\text{sht}}$ | $\mathbf{z}_{\text{ssd}}$ | $\mathbf{z}_{\text{con}}$ | $\mathbf{z}_{\text{sht}}$ |
|---|---|---|---|---|---|---|---|---|---|
| jagged-dubin | 59.2 | 90.2 | 82.6 | 82.2 | 74.7 | 82.3 | 82.2 | 70.1 | 63.15 |
| gap-dubin | 56.4 | 90.5 | 88.6 | 87.9 | 77.3 | 86.2 | 87.1 | 76.2 | 76.0 |
| squared-dubin | 56.8 | 98.3 | 89.3 | 87.1 | 77.5 | 88.2 | 85.0 | 61.7 | 61.5 |

TABLE I

EFFECTIVENESS PREDICTION ACCURACY (IN %) FOR THE DUBINS CAR PLANNING: WE COMPARE THE PREDICTION ACCURACY OF THE BASELINE (I.E., THE PERCENTAGE OF MAJORITY TRAJECTORY LABEL) WITH THE PREDICTION RESULTS WHEN USING ALL THE FEATURES WE DESIGNED $\mathbf{z}_{\text{CONV}}+\mathbf{z}_{\text{SSD}}+\mathbf{z}_{\text{CON}}+\mathbf{z}_{\text{SHT}}$ AND OTHER COMBINATIONS OF FEATURES.

| | baseline | $\mathbf{z}_{\text{conv}}+\mathbf{z}_{\text{ssd}}+\mathbf{z}_{\text{con}}+\mathbf{z}_{\text{sht}}$ | $\mathbf{z}_{\text{ssd}}+\mathbf{z}_{\text{con}}+\mathbf{z}_{\text{sht}}$ | $\mathbf{z}_{\text{ssd}}+\mathbf{z}_{\text{con}}$ | $\mathbf{z}_{\text{con}}+\mathbf{z}_{\text{sht}}$ | $\mathbf{z}_{\text{ssd}}+\mathbf{z}_{\text{sht}}$ | $\mathbf{z}_{\text{ssd}}$ | $\mathbf{z}_{\text{con}}$ | $\mathbf{z}_{\text{sht}}$ |
|---|---|---|---|---|---|---|---|---|---|
| bookshelf | 63.0 | 100 | 77.7 | 77.5 | 73.9 | 77.5 | 76.3 | 68.7 | 70.0 |
| skew-bookshelf | 62.1 | 85.9 | 80.4 | 74.3 | 77.9 | 76.2 | 72.3 | 63.7 | 65.0 |
| countertop | 69 | 99.7 | 70.1 | 72.8 | 74.5 | 69.6 | 73.9 | 77.7 | 63.6 |
| industrial | 67.2 | 83.8 | 78.5 | 78.2 | 78.4 | 78.2 | 76.9 | 80.2 | 73.7 |
| industrial2 | 50.3 | 96.2 | 61.1 | 62.1 | 64.1 | 61.2 | 62.3 | 64.5 | 58.2 |

TABLE II

EFFECTIVENESS PREDICTION ACCURACY (IN %) FOR PR2 ARM PLANNING: WE COMPARE THE PREDICTION ACCURACY OF THE BASELINE (I.E., THE PERCENTAGE OF MAJORITY TRAJECTORY LABEL) WITH THE PREDICTION RESULTS WHEN USING ALL THE FEATURES WE DESIGNED $\mathbf{z}_{\text{CONV}}+\mathbf{z}_{\text{SSD}}+\mathbf{z}_{\text{CON}}+\mathbf{z}_{\text{SHT}}$ AND OTHER COMBINATIONS OF FEATURES.

| | baseline | $\mathbf{z}_{\text{ssd}}+\mathbf{z}_{\text{con}}+\mathbf{z}_{\text{sht}}$ | $\mathbf{z}_{\text{ssd}}+\mathbf{z}_{\text{con}}$ | $\mathbf{z}_{\text{con}}+\mathbf{z}_{\text{sht}}$ | $\mathbf{z}_{\text{ssd}}+\mathbf{z}_{\text{sht}}$ | $\mathbf{z}_{\text{ssd}}$ | $\mathbf{z}_{\text{con}}$ | $\mathbf{z}_{\text{sht}}$ |
|---|---|---|---|---|---|---|---|---|
| livingroom | 61.0 | 85.5 | 88.4 | 83.6 | 84.7 | 87.4 | 82.8 | 81.6 |
| livingroom2 | 53.7 | 80.3 | 80.5 | 76.7 | 78.6 | 77.9 | 71.5 | 66.6 |
| kitchen | 59.0 | 65 | 64.2 | 60.4 | 66.9 | 66.2 | 60.3 | 73.6 |
| kitchen2 | 54.0 | 64.6 | 65.3 | 61.6 | 65.0 | 65.0 | 61.0 | 61.5 |
| hotel | 98 | 98.1 | 98.1 | 98.2 | 98.1 | 98.1 | 97 | 98.1 |

TABLE III

EFFECTIVENESS PREDICTION ACCURACY (IN %) FOR PR2 FULLBODY PLANNING: WE COMPARE THE PREDICTION ACCURACY OF THE BASELINE (I.E., THE PERCENTAGE OF MAJORITY TRAJECTORY LABEL) WITH THE PREDICTION RESULTS WHEN USING FEATURES $\mathbf{z}_{\text{SSD}}+\mathbf{z}_{\text{CON}}+\mathbf{z}_{\text{SHT}}$ AND OTHER COMBINATIONS OF FEATURES.

[22] S. Cameron, "Enhancing GJK: computing minimum and penetration distances between convex polyhedra," in *Proceedings of International Conference on Robotics and Automation*, vol. 4, 1997, pp. 3112–3117.

[23] E. Gilbert, D. Johnson, and S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal of Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[24] G. Bergen, "Proximity queries and penetration depth computation on 3d game objects," in *Game Developers Conference*, 2001.

[25] I. Sucan, M. Kalakrishnan, and S. Chitta, "Combining planning

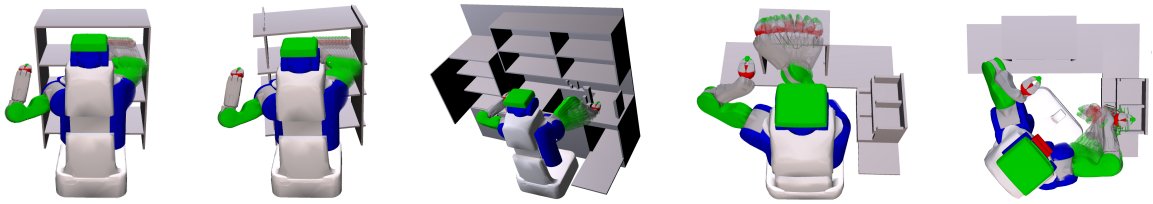
| | baseline | $\mathbf{z}_{\text{conv}}+\mathbf{z}_{\text{ssd}}+\mathbf{z}_{\text{con}}+\mathbf{z}_{\text{sht}}$ | $\mathbf{z}_{\text{ssd}}+\mathbf{z}_{\text{con}}+\mathbf{z}_{\text{sht}}$ | $\mathbf{z}_{\text{ssd}}+\mathbf{z}_{\text{con}}$ | $\mathbf{z}_{\text{con}}+\mathbf{z}_{\text{sht}}$ | $\mathbf{z}_{\text{ssd}}+\mathbf{z}_{\text{sht}}$ | $\mathbf{z}_{\text{ssd}}$ | $\mathbf{z}_{\text{con}}$ | $\mathbf{z}_{\text{sht}}$ |
|---|---|---|---|---|---|---|---|---|---|
| jagged-dubin to squared-dubin | 56.8 | 88.8 | 49.7 | 49.0 | 74.4 | 56.9 | 60.3 | 67.9 | 55.5 |
| squared-dubin to jagged-dubin | 59.2 | 58.9 | 56.9 | 56.8 | 71.1 | 56.9 | 56.9 | 61.2 | 63.8 |
| bookshelf to skew-bookshelf | 62.1 | 86.7 | 57.1 | 59.4 | 82.1 | 53.2 | 57.3 | 60.3 | 67.4 |

TABLE IV

EFFECTIVENESS PREDICTION ACCURACY (IN %) WHEN APPLYING THE CLASSIFIER LEARNED ON ONE BENCHMARK ONTO TRAJECTORIES FROM OTHER BENCHMARKS. THE BASELINE IS THE PERCENTAGE OF MAJORITY TRAJECTORY LABEL OF THE BENCHMARK WHERE THE CLASSIFIER IS TRANSFERRED TO.

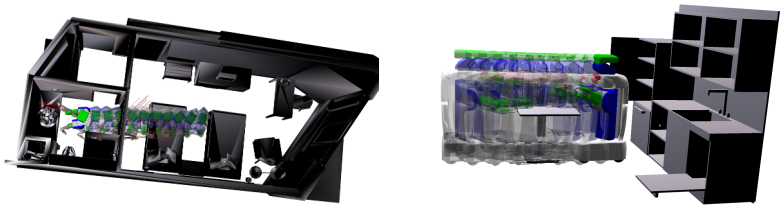
(a) jagged-dubin (b) gap-dubin (c) squared-dubin

Fig. 6. The dubins car environment for 2D experiments

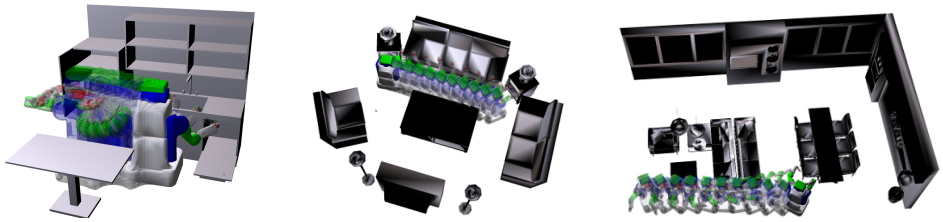


(a) bookshelf (b) skew-bookshelf (c) countertop (d) industrial (e) industrial2

Fig. 7. The PR2 benchmarks for right arm planning with 22 links.



(a) hotel (b) kitchen



(c) kitchen2 (d) livingroom (e) livingroom2

Fig. 8. The PR2 benchmarks for full-body planning with 88 links.

techniques for manipulation using realtime perception," in *Proceedings of International Conference on Robotics and Automation*, 2010, pp. 2895–2901.

[26] A. Yershova, S. Jain, S. M. Lavalle, and J. C. Mitchell, "Generating uniform incremental grids on so(3) using the hopf fibration," *International Journal of Robotics Research*, vol. 29, no. 7, pp. 801–812, June 2010.

[27] A. Frome, D. Huber, R. Kolluri, T. Blow, and J. Malik, "Recognizing objects in range data using regional point descriptors," in *Proceedings of European Conference on Computer Vision*, 2004, pp. 224–237.

[28] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz, "Rotation invariant spherical harmonic representation of 3d shape descriptors," in *Proceedings of Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, 2003, pp. 156–164.

[29] C. Cortes and V. Vapnik, "Support-vector networks," *Maching Learning*, vol. 20, no. 3, pp. 273–297, 1995.