Noname manuscript No.
(will be inserted by the editor)

Fast Motion Planning from Experience: Trajectory Prediction for Speeding up Movement Generation

Nikolay Jetchev · Marc Toussaint

Received: date / Accepted: date

Abstract Trajectory planning and optimization is a fundamental problem in articulated robotics. Algorithms used typically for this problem compute optimal trajectories from scratch in a new situation, without exploiting similarity to previous problems. In effect, extensive data is accumulated containing situations together with the respective optimized trajectories—but this data is in practice hardly exploited. This article describes a novel method to learn from such data and speed up motion generation, a method we denote Trajectory Prediction.

The main idea is to use demonstrated optimal motion trajectories to quickly predict appropriate trajectories for novel situations. These can be used to initialize and thereby drastically speed-up subsequent optimization of robotic movements and improve the convergence behavior of a conventional motion optimizer. Our approach has two essential ingredients. First, to generalize from previous situations to new ones we need an appropriate situation descriptor – we construct features for such descriptors and use a sparse regularized feature selection approach to find well-generalizing features of situations. Second, the transfer of previously optimized trajectories to a new situation should not be made in joint angle space – we propose a more efficient task space transfer of old trajectories to new situations.

We present extensive results in simulation to illustrate the benefits of the new method, and demonstrate it also with real robot hardware. Our experiments with a reaching and obstacle avoidance task, and an object grasping task, show that we can predict good motion trajectories in new situations for which the refinement is much faster than an optimization from scratch.

Keywords motion planning · machine learning · motion representation · articulated robotics

Nikolay Jetchev and Marc Toussaint are with Machine Learning and Robotics Lab, FU Berlin, Arnimallee 7, 14195 Berlin, Germany
E-mail: nikolay.jetchev@fu-berlin.de, marc.toussaint@fu-berlin.de

1 Introduction

Motion planning is a fundamental issue in articulated robotics. It is a crucial component for many tasks, the most basic of which, reaching to a target location without hitting obstacles, will be used as example application in this work. This article describes a method that can speed up motion planning by improving the initialization used in stochastic optimal control planners. This is a sensitive aspect of such local planners: they can fall in multiple local optima and a good solution is not guaranteed. Using the structure of encountered environments can provide hints about movements that are likely to be good in a given world configuration.

The animal and human ability to generate trajectories quickly is amazing. In typical every-day situations humans do not seem to require time for motion planning but execute complex trajectories instantly. This suggests that there exists a "reactive trajectory policy" which maps "the situation" (or at least motion relevant features of the situation) to the whole trajectory.[1] Such a mapping (if optimal) is utterly complex: the output is not a single current control signal but a whole trajectory which, traditionally, would be the outcome of a computationally expensive trajectory optimization

[1] This is not to be confused with a reactive controller which maps the current sensor state to the current control signal—such a (temporally local) reactive controller could not explain trajectories which efficiently circumvent obstacles in an anticipatory way, as humans naturally do in complex situations.

process accounting for collision avoidance, smoothness and other criteria. The input is the current situation, in particular the position of relevant objects, for which it is unclear which representation and coordinate systems to use as a descriptor.

The goal of the current work is to learn such an (approximate) mapping from data of previously optimized trajectories in old situations to good trajectories in new situations. We coin this problem *Trajectory Prediction* (TP). In Section 2 we will examine the basics of motion planning as optimization task, and give an overview of TP and how it is coupled with planning. Afterwards we will proceed with Section 3 where we examine what representations of states allow efficient generalization of movements between situations. In Section 4 we will present the Inverse Kinematics (IK) Transfer operator we use, and discuss what task space representations are appropriate for such transfer. The way we learn a policy used for predicting situation-appropriate trajectories will be explained in Section 5. In Section 6 we will discuss connections between TP and imitation learning. Finally, in the experimental Section 7 we will show our results for several simulated robot motion planning scenarios.

The main contributions of our work can be summarized as follows:

- the TP method for speeding up planning by learning a predictive model of motions appropriate to be transferred to a new situation
- the definition of representations that allow accurate mapping of situation to movement
- the notion of IK Transfer in task space, allowing robust generalization of the predicted movements between different situations
- quantitative results in three different motion planning tasks showing how TP can speed up motion planning

We finish the current introduction with a brief overview of relevant motion planning and learning methods.

1.1 Related Motion and Trajectory Generation Methods

1.1.1 Local Planning Methods

Movement generation, one of the most basic robotic tasks, is often viewed as an optimization problem that aims to minimize a cost function. There are many different methods for local trajectory optimization which use the cost gradient information for minimization. Popular approaches use spline-based representation and gradient descent (Zhang and Knoll 1995), covariant gradient descent (Ratliff et al 2009), Differential Dynamic Programming (DDP) described by Dyer and McReynolds (1970); Atkeson (1993), a variant of DDP called iterated Linear Quadratic Gaussian (iLQG) by Todorov and Li (2005), and Bayesian inference (Toussaint 2009). Such methods are usually fast and can obtain movements of good quality, suitable for control of complex hardware robots with many DoF. However, these local methods can get stuck in local optima. TP aims to predict directly good trajectories such that local planners only need to refine them.

1.1.2 Rapidly-exploring Random Trees (RRT) and Other Sampling Methods

Another approach for finding good movement trajectories is sampling to find obstacle free paths in the configuration and work space of the robot, i.e. finding an appropriate initialization of the movement plan. Popular methods for planning feasible paths without collisions are RRTs Bertram et al (2006) and probabilistic road maps Kavraki et al (1995), where random sampling is used to build networks of feasible configuration nodes. These methods are powerful and can find difficult solutions for motion puzzles, but also have the disadvantage to be too slow for high-dimensional manipulation problems. Building an RRT takes some time, and a path to the target in such a network often requires additional optimization to derive an optimal robot trajectory. In contrast, TP is much faster in providing an initial motion, and is designed also to work well in conjunction with a motion planner for refinement.

1.2 Previous Use of Machine Learning Techniques to Speed up Planning

1.2.1 Transfer in Reinforcement Learning

Concerning our problem of learning from previous optimization data, there exist multiple branches of related work in the literature. In the context of Reinforcement Learning the transfer problem has been addressed, where the value function (Konidaris and Barto 2006) or directly the policy (Peshkin and de Jong 2002) is transferred to a new Markov Decision Process. Konidaris and Barto (2006) discussed the importance of representations for the successful transfer. Although the problem setting is similar, these methods are different in that they do not consider a situation descriptor (or features of the "new" MDP) as an input to a mapping which directly predicts the new policy or value function.

1.2.2 Robot Motion Databases and Learning from Demonstration

Related work with respect to exploiting databases of previous trajectories has been proposed in the context of RRTs. Branicky et al (2008) constructed a compact database of collision free paths that can be reused in future situations to speed up planning under the assumption that some of the previous paths will not be blocked by future obstacles and can be reused for fast planning. Martin et al (2007) attempted to bias RRTs such that after planning in a set of initial environments, the obstacles can be rearranged and previous knowledge will be used for faster replanning in the new scene; an environment prior, that visits with higher probability states visited in previous trials, is used to speed up planning and use less tree nodes to achieve the final goal. In both cases, the notion of our situation descriptor and the direct mapping to an appropriate new trajectory is missing.

Another interesting way to exploit a database of previous motions is to learn a "capability map", i.e., a representation of a robot's workspace that can be reached easily, see Zacharias et al (2007). While this allows to decide whether a certain task position can be reached quickly, it does not encode a prediction of a trajectory in our sense.

Stolle and Atkeson (2007) predict robot locomotion movements for navigation in new situations using databases of state-action pairs to make small steps ahead. In a sense, such use of a database presents action primitives extracted from data similar to TP. However, unlike TP, Stolle and Atkeson (2007) adapt their algorithm specifically to the locomotion navigation domain by combining local step planning with global graph-based search, and does not learn data-driven situation feature representations.

The field of imitation learning (Argall et al 2009) encompasses many approaches using demonstrated motions to learn behaviors: policies that map from situations to actions. The focus is usually to extract motions from human demonstration of different tasks which can be later repeated "exactly" by robots, e.g. see Calinon and Billard (2005); Shon et al (2007) .The demonstrations, often complex gestures or manipulations, are to be repeated accurately, possibly with some robustness to perturbation. However, generalization to different environments and collision avoidance with obstacles there is rarely considered in the imitation process. This is not surprising, since acquiring data in an interactive way is costly and limits the variation of situations and motions that can be encountered. In Section 6 this comparison between TP and imitation learning will be discussed in more detail.

TP approaches motion planning problems in a framework to improve the convergence of local motion planners by predicting situation-appropriate motions. TP predicts whole trajectories at once, not requiring additional global search routines. It seems reasonable that good paths will go around obstacles, and TP can potentially provide a way to start the motion planning task with a path avoiding collisions, similar to RRT. However, our prediction method will not be limited to obstacle avoidance only: TP will predict motion trajectories that improve the convergence of local planners and deals with all aspects implicit in a low cost.

2 Planning Motion and Predicting Motion

We assume that the desired behavior of the robot is to generate a motion trajectory good for some specified task. As mentioned in the introduction, such a desired trajectory can be calculated by a planner module minimizing a cost function. One can think of the behavior of such a planner (and some heuristic for initialization) as a policy mapping a situation x to a joint trajectory $\boldsymbol{q}$. We propose to use experience in the form of demonstrated optimal trajectories in different situations as initialization for local planners, resulting in a better movement policy. This section will proceed by first describing the planning by cost function motion model and then formalizing TP.

2.1 Robot Motion Planning: a Basic Model

Let us describe the robot configuration at time t as $q_t \in \mathbb{R}^N$, the joint posture vector. We define $\boldsymbol{q} = (q_0, .., q_T)$ as a movement trajectory with time horizon of T steps. In a given situation x, i.e., for a given initial posture q_0 and the positions of obstacle and target objects in this problem instance (we will formally define descriptors for x in Section 3), a typical motion planning problem is to compute a trajectory which fulfills different constraints, e.g. an energy efficient movement not colliding with obstacles. We formulate this as an optimization problem by defining a cost function

$$C(x, \boldsymbol{q}) = \sum_{t=1}^{T} g_t(q_t) + h_t(q_t, q_{t-1}) \ . \quad (1)$$

that characterizes the quality of the joint trajectory in the given situation and task constraints. We will specify such a cost function explicitly in our experiments section. Generally, g will account for task targets and collision avoidance, and h for control costs.

A trajectory optimization algorithm essentially tries to map a situation x to a trajectory $\boldsymbol{q}$ which is optimal,

$$x \mapsto \boldsymbol{q}^* = \underset{\boldsymbol{q}}{\operatorname{argmin}}\, C(x, \boldsymbol{q}) \ . \tag{2}$$

For this we assume to have access to $C(x, \boldsymbol{q})$ and local (linear or quadratic) approximations of $C(x, \boldsymbol{q})$ as provided by a simulator, i.e., we can numerically evaluate $C(x, \boldsymbol{q})$ for given x and $\boldsymbol{q}$ but we have no analytic model. To arrive at the optimal trajectory $\boldsymbol{q}^*$ (or one with a very low cost C), most local optimizers start from an initial trajectory $\tilde{\boldsymbol{q}}$ and then improve it. We call $\mathcal{O}$ the local optimizer operator and write $\boldsymbol{q}^* = \mathcal{O}_x(\tilde{\boldsymbol{q}})$ when we optimize for a specific situation x.

Optimizing C is a challenging high-dimensional non-linear problem. Many of the movement optimization methods are sensitive to initial conditions and their performance depends crucially on it. For example, initial paths going straight through multiple obstacles are quite difficult to improve on, since the collision gradients provide confusing information and try to jump out of collision in different conflicting directions, as mentioned by Ratliff et al (2009). RRT and sampling methods for motion planning are used for finding good paths, but with the drawback of higher computational burden: construction of tree with collision free steps and subsequent dynamic optimization of a whole trajectory from one path in the tree.

2.2 Trajectory Prediction: Definition and Overview of our Algorithm

In this section we first define the trajectory prediction problem in general terms and outline how we break down the problem in three steps: (i) finding appropriate task space descriptors, (ii) transfer of motion prototypes to new situations, and (iii) learning a predictive model of which motion prototype is appropriate to be transferred to a new situation.

The goal of TP is to learn an approximate model of the mapping (2) from a data set of previously optimized trajectories. The dataset D comprises pairs of situations and optimized trajectories,

$$D = \{(x_i, \boldsymbol{q}_i)_{i=1}^d\} \ , \quad \boldsymbol{q}_i \approx \underset{\boldsymbol{q}}{\operatorname{argmin}}\, C(x_i, \boldsymbol{q}) \ . \tag{3}$$

The full sequence involved in TP, which we will explain below, is the following:

$$x \to \hat{i} \to \mathcal{T}_{x x_{\hat{i}}} \boldsymbol{q}_{\hat{i}} \to \mathcal{O}_x \mathcal{T}_{x_{\hat{i}} x} \boldsymbol{q}_{\hat{i}} = \boldsymbol{q}^* \tag{4}$$

TP takes as input an appropriately represented situation descriptor x, see Section 3. We then predict the index $\hat{i}$ of a motion from D to be executed and transfer it with the operator $\mathcal{T}$ from situation $\hat{i}x$ to x, described in Section 4. We can view the subsequence

$$f : x \to \mathcal{T}_{x_{\hat{i}} x} \boldsymbol{q}_{\hat{i}} \tag{5}$$

as the policy mapping situation to motion, and will explain it in Section 5. Finally, the above TP sequence ends with applying the planning operator $\mathcal{O}_x$. Prediction without any subsequent optimization would correspond to pure imitation, and our method is not designed with such aim. TP is inherently coupled with a planner that minimizes the cost function C, so the prediction policy is designed to speed-up such a planner.

As an aside, this problem setup generally reminds of structured output regression. However, in a structured output scenario one learns a discriminative function $C(x, \boldsymbol{q})$ for which $\underset{\boldsymbol{q}}{\operatorname{argmin}}\, C(x, \boldsymbol{q})$ can efficiently be computed, e.g. by inference methods. Our problem is quite the opposite: we assume $\underset{\boldsymbol{q}}{\operatorname{argmin}}\, C(x, \boldsymbol{q})$ is very expensive to evaluate and thus learn from a data set of previously optimized solutions. A possibility to bring both problems together is to devise approximate, efficiently computable structured models of trajectories and learn the approximate mapping in a structured regression framework. But this is left to future research.

In the next sections we will continue with detailed description of the elements of TP.

3 Situation Representations and Descriptor

A typical scenario for articulated motion generation is a workspace filled with objects and a robot. A situation (or problem instance) is fully specified by the initial robot posture q_0 and the positions of obstacles and targets in this problem instance. There are a lot of possible features we can construct to capture relevant situation information. For instance, positions of obstacles could be given relative to some coordinate system in the frame of some other object in the scene. We should expect that our ability to generalize to new situations crucially depends on the representations we use to describe situations.

We present two different approaches for modeling x, appropriate for scenarios with different assumptions. This section will proceed by describing these two models.

3.1 General Geometric Descriptor

Our first approach is to define a very high-dimensional and redundant situation descriptor which includes distances and relative positions w.r.t. many different frames

of reference. Training the predictive function then includes selecting the relevant features. Assume we have a set of b different 3D objects (i.e. landmarks) in the scene which might be relevant for motion generation: $A = (a_1, .., a_b)$ with each $a_j \in \mathbb{R}^3$. We create features by examining the geometric relations between pairs of such objects. For b such landmarks we have $\hat{b} = b(b-1)$ such pairs. For each pair $i = (i_1, i_2) \in \{1, ..., \hat{b}\}$ we measure the 3D relative difference between landmarks in A in the frame of a_{i_2} as $p_i = (p_i^x, p_i^y, p_i^z)$ and its norm $d_i = \|p_i\|$. We also define the azimuths of the three axes as $\psi_i = \{\arccos(p_i^x/d_i), \arccos(p_i^y/d_i), \arccos(p_i^z/d_i)\}$. We gather this basic geometric information in the 7 dimensional vector $\phi_i = (p_i, d_i, \psi_i)$. The final descriptor x comprises all these local pairwise vectors:

$$x = (\phi_1, ..., \phi_{\hat{b}}) \in \mathbb{R}^{7\hat{b}} \tag{6}$$

Even more complex geometric descriptors are possible, but the choice of (6) turned out to be sufficient in our experiments. Given such a descriptor we can use a feature selection technique to infer from the data which of these dimensions are best for trajectory prediction in new situations. In the experimental Section 7 we will show how extracting a sparse representation from this redundant description provides an interesting explanation of the important factors in a situation giving rise to different motions.

3.2 Voxel Descriptor

The approach to model directly the distances between object centers is appropriate for situations with few obstacles with simple geometries, but it can have issues with scaling when more objects are present. We also present an extension appropriate for cluttered scenes where obstacles are modeled from point clouds of 3D sensor data, which can handle multiple objects easily. We call this a sensor driven approach to trajectory prediction. Since modeling each of these as an object with coordinates in the descriptor x is impractical, we use here voxel information to make the descriptor x.

We assume that a sensor (LIDAR or stereovision) is available that provides information in the form of a point cloud from detected objects, which can be then converted to a voxel representation of a scene, see Elfes (1989); Nakhaei and Lamiraux (2008). This information representing the obstacles is crucial for the correct task execution, an assumption appropriate for cluttered scenes and navigation. Given a set of laser cloud points $P = \{p_i\}$, we construct a 3D grid system $V = \{v_i\}$ of voxels. Each voxel is identified with its coordinates and its occupancy probability $p(v_i) \in [0, 1]$. The procedure for calculating $p(v)$ is straightforward:

1. Loop through all available measurements p_i
2. Loop through all voxels v_j
3. If $p_i \subset v_j$ set $p(v_j) = 1 - 0.9 * (1 - p(v_j))$

The idea is that for every measurement point within some voxel bounds the occupied space probability of the voxel increases. Elfes (1989) and Nakhaei and Lamiraux (2008) use sensor models with state distributions for free, unknown and occupied voxel space, but for our tests only the occupied space probability model suffices for collision avoidance.

To better explain the voxel descriptor, we will describe how it will look concretely in our experiments. We define two such voxel grids, 15 voxels across each dimension, where each voxel is a cube with side 7cm. The first grid is centered at the center of the workspace, the second on the target location. Each voxel grid V has can be described as a vector of dimension $15^3 = 3375$ containing the values of all its cells $p(v_i)$. We can compress a voxel grid using standard Principal Component Analysis (PCA) to the 200 most significant dimensions, and thus have the following grid descriptor $v = VP \in \mathbb{R}^{200}$ where P is the projection matrix calculated by PCA. By taking only the columns of P with highest variance one can get a relatively good variance preserving compression of the voxel grids. In Figure 1 we show what is the result of the PCA transformation used on the voxel grid data. The PCA coefficients for each component, the column vectors of the PCA projection matrix P which have the same dimension as the uncompressed voxel grids, can be interpreted as characteristic terrains of the voxel grid. Larger values indicate larger probability that a terrain is occupied. Note that the PCA decomposition for voxel data is useful to discriminate between world configurations and represent general notions like whether the left or right side of the workspace is free. A lot of the detailed voxel information about the world is lost by PCA, but TP is more efficient with lower dimensional descriptors.

The final situation descriptor is than:

$$x = \{d, v_1, v_2\} \in \mathbb{R}^{413} \tag{7}$$

The entries v_1, v_2 are the two PCA compressed voxel grid descriptors, and $d \in \mathbb{R}^{13}$ contains additional scene information, the initial 7D robot arm joint position, the 3D endeffector position and target position.

4 Task Space Trajectory IK Transfer

In this section we will describe the exact way in which we repeat and adapt a motion from the database to a random new situation.

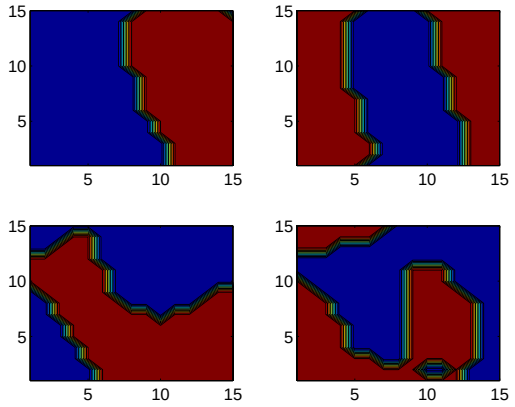


Fig. 1 The first 4 PCA components, visualized in the square plain of size 15x15 voxels from a slice 7cm above the center of the grid. Red areas are more likely to be occupied, and blue areas are probably free.

4.1 Motion Representation for Output Trajectory Task Space

As we mentioned in section 2 we will motion trajectories transferred via some task space. The projection $\boldsymbol{y}$ of a trajectory $\boldsymbol{q}$ into task space is defined as $\boldsymbol{y} = \phi_x(\boldsymbol{q})$, where ϕ_x is a kinematic mapping (depending on the situation x) applied to each time slice, with a task space for output.

Some obvious choices of task spaces are the joint angle space Q (mapped by identity) and Y, the space of world coordinates of robot hand endeffector (mapped by the hand kinematics). However, these have the drawback of not generalizing well - a simple change in a world situation like translation of some object would make a movement prototype in such space unfeasible for the changed situation. A reasonable choice of task space can ensure at least some degree of generalizing ability in a new situation. For example, we would also consider the task space Y_{target} of coordinates starting in a world frame centered on the target, and Y_{obst}, a world frame starting in the center of the largest obstacle in the scenes we examined.

Our current approach to task space selection is to test empirically task spaces that seem reasonable, and to select the space that allows the best planner initialization. This is a similar approach to Muehlig et al (2009).

The question of what are suitable representations of a physical configuration, in particular suitable coordinate systems, has previously been considered in a number of works. Wagner et al (2004) discussed the advantages of egocentric versus allocentric coordinate systems for robot control, and Hiraki et al (1998) talked about such coordinates in the context of robot and human learning.

4.2 The Transfer Operator

Suppose we want to transfer the joint motion $\boldsymbol{q}'$ which was optimal for situation x'. A task space trajectory $\boldsymbol{y} = \phi_{x'}(\boldsymbol{q}')$ needs to be transformed back to a joint space trajectory $\boldsymbol{q}$ in order to initialize the local motion planner in a new situation x, see the sequence of Equation (4). Simply repeating the old motion with IK is likely to be problematic, e.g. motion targets and obstacles change between situations. Therefore we use IK with multiple task variables (cost terms from the planning cost function from Equation 1) to transfer motions and adapt to new situations. We generate a motion for each time slice $t = 1...T$ by iteratively finding the next steps

$$q_t = \phi_x^{-1}(q_{t-1}, q'_t) = \underset{q}{\operatorname{argmin}}\, C^{\text{IK}}(x, q, q_{t-1}, q'_t) \tag{8}$$

This next-step cost C^{IK} is defined as

$$C^{\text{IK}}(x, q_t, q_{t-1}, q'_t) = g(q_t) + h(q_t, q_{t-1}) + \|\phi_x(q_t) - \phi_{x'}(q'_t)\|^2 \tag{9}$$

where q'_t is the motion being transferred, and q_t are time slices of the old motion being transferred $\boldsymbol{q}'$ and the result of this transfer $\boldsymbol{q}$.

In effect, $\phi_x^{-1}(q_{t-1}, q'_t)$ roughly follows in the current state q_{t-1} the next state of the transferred motion $\phi_{x'}(q'_t)$ in task space. The terms h and g ensure that the next steps correct for collisions and other terms of the cost function from Equation (1), fitting the movement to the new situation. The mapping $\phi_x^{-1} : (\boldsymbol{y}, q_0) \mapsto \boldsymbol{q}$ projects the whole task space trajectory to a joint space trajectory in situation x, applying the IK operator ϕ_x^{-1} for each time step t, starting from q_0.

The transfer operator is then the following function composition:

$$\mathcal{T}_{xx'}\boldsymbol{q}' = \phi_x^{-1} \circ \phi_{x'}(\boldsymbol{q}') \tag{10}$$

Such a transfer gives our method better generalization ability, since a motion which by itself is not optimal for the motion task can still be followed in the new situation will be followed and adapted with IK, and can still lead to a good initialization.

For an illustration of the IK mapping ϕ_x^{-1} consider a simplified example. Assume that $h(q_t, q_{t-1}) = \|q_t - q_{t-1}\|^2_W$, and $g(q_t) = \|\varphi_x(q_t) - \varphi^*_x\|^2_C$ where φ_x is another kinematic function with targets φ^*. The diagonal matrices C and W give the precision (i.e. weighting) of the different cost terms. We linearize $\phi_x(q_t) =$

$\phi_x(q_{t-1}) + J_1(q_t - q_{t-1})$ where $J_1 = \frac{\delta \phi_x(q)}{\delta q}$ is the Jacobian. Similarly, $\varphi_x(q_t) = \varphi_x(q_{t-1}) + J_2(q_t - q_{t-1})$ and $J_2 = \frac{\delta \varphi_x(q)}{\delta q}$. The optimal solution of Equation (8) with this linearization is

$$\begin{aligned} \hat{J} &= [J_1^T J_1 + J_2^T C J_2 + W]^{-1} \\ q_t &= q_{t-1} + \hat{J}[J_1^T(\phi_{x'}(q'_t) - \phi_x(q_t)) \\ &\quad + J_2^T C(\varphi^* - \varphi_x(q_t))]. \end{aligned} \tag{11}$$

5 Mapping Situation to Motion

Once we have defined appropriate situation descriptors and the IK transfer method of adapting motions from one situation to another, we can proceed to describe the mapping f predicting "situation-appropriate" movements that lead to quick motion planner convergence.

5.1 Gathering Data Demonstrating Behavior

The first step toward learning f is the gathering of optimal trajectories in different random situations. The dataset D comprises pairs of randomly generated situations x and trajectories $\boldsymbol{q}$, optimized offline to convergence with a local planner with default initialization:

$$D = \{(x_i, \boldsymbol{q}_i)_{i=1}^d\} \tag{12}$$

Allowing the local planner to run for lots of iterations and time makes getting an optimal trajectory very likely, but failure is still possible, as our results will indicate later. For the set D we retained only good movements and discarded the failed attempts.

Then we can gather data D' for the quality of the initialization using the new actions in different situations, and how much additional refinement they need from the planner. We measure this "refinement cost" as

$$F(x, \boldsymbol{q}) = C(x, \mathcal{O}_x^j \boldsymbol{q}) \tag{13}$$

Here $\mathcal{O}_x^j \boldsymbol{q}$ is the trajectory vector found by the optimizer after j iterations, starting from initialization $\boldsymbol{q}$, and j is a constant. Such a definition of F is a heuristic to quantify the effect of initialization on convergence speed looking only at few planner iterations, which is possible because the planners we use iteratively improve the solution making small steps.

The cross initialization dataset D' is defined as:

$$D' = \{(x_j, x_i, F(x_j, \mathcal{T}_{x_j x_i} \boldsymbol{q}_i)\}\ , \quad x_j \in D^x, (x_i, \boldsymbol{q}_i) \in D \tag{14}$$

That is, we evaluate the quality of initialization in situation x_j of a database movement $\boldsymbol{q}_i$ transferred from x_i, and this is the data we will use to learn a good mapping f. The set D^x has a new set of situations where we examine the cost of transferred motions from set D.

We can examine potentially the effect on convergence speed of every optimal movement demonstrated in the set D, but in the experiments section we will also test using smaller representative sets of motions (e.g. by using clustering) to select smaller subsets of D with different motion types. This would be a compression of the trajectory sets allowing to evaluate a smaller number of costs for initializations.

Creating D requires $d = |D|$ planner calls until convergence to a local optima. Creating D' requires $|D^x|d$ planner calls, each of which takes j iterations, and can be made faster by running for fewer iterations j to evaluate F.

A difference between the datasets D and D' is due to the different initializations used to create them. For the set D we use the planners with default initialization without experience of previous situations as a module to get optimal movements for the different situations, and retain only the successful runs. In the second dataset D' we use examples of good motions from set D, and use IK transfer to adapt the trajectories(actions) for better generalization between situations. We measure the convergence cost with the measure F for a limited amount of iterations.

Once we have gathered data in the set D' as defined in Equation (14), we can use it to learn the trajectory prediction mapping from Equation (5). This is a supervised learning problem, and the next subsections describe two possible approaches to learning the mapping f. As preprocessing for all prediction methods, we rescale each dimension of the descriptors x in [0,1] by subtracting the minimum and rescaling, which improves performance of prediction methods.

5.2 Nearest Neighbor Predictor

We assume we start with a descriptor vector x, which is potentially redundant and high dimensional. We assume that similar situations have similar optimal trajectories. However, the usual notion of similarity as the negative Euclidean distance may not be the best for the high dimensional situation descriptors we have defined. We want to learn a similarity metric w in the situation descriptor feature space that selects appropriate features. Our learning method will allow to retain the most representative and compact dimensions, in addition to improving the trajectory prediction quality.

We define the situation similarity function as:

$$k(x, x_i) = \exp\{-\frac{1}{2}(x - x_i)^T W (x - x_i)\} \quad (15)$$

$$W = \text{diag}(w_1^2, .., w_s^2) \ , \quad (16)$$

The nearest neighbor predictor f for x is

$$f(x) = \mathcal{T}_{xx_{\hat{i}}} \boldsymbol{q}_{\hat{i}} \ , \quad \hat{i} = \underset{i \in D}{\text{argmax}} \ k(x_i, x) \quad (17)$$

The probability to choose a specific trajectory $i \in D$ with such similarity is:

$$P(f(x) = \mathcal{T}_{x_i x} \boldsymbol{q}_i) = \frac{1}{Z} k(x, x_i) \quad (18)$$

with $Z = \sum_{i \in D} k(x, x_i)$ as normalizing constant.

We can define the expectation over the planner costs in situation x when initializing with Equation 18 as:

$$\text{E}\{F(x, f(x))\} = \sum_{i \in D} P(f(x) = \mathcal{T}_{xx_i} \boldsymbol{q}_i) F(x, \mathcal{T}_{xx_i} \boldsymbol{q}_i) \quad (19)$$

Our goal is to find a similarity metric with low expected motion planning costs over all situations for which we have convergence information. We define the following loss function L using the cross initialization data D':

$$L(w; D') = \frac{1}{|D^x|} \sum_{x \in D^x} \text{E}\{F(x, f(x))\} + \lambda |w|_1 \quad (20)$$

By minimizing this loss function we do feature selection to improve the similarity metric used for nearest neighbor classification. The purpose of the L_1 regularization is to get sparse similarity metrics using only few situation features.

Learning a similarity metric that describes well which situations have movements suitable for transfer has an interesting property: we transfer knowledge of expected costs for yet unseen movements, an action set of potentially unlimited size. We will examine this in the experiment section.

5.3 Trajectory Prediction via SVR

As an alternative to the above prediction scheme and for empirical evaluation we also test a multi-linear Support Vector Regression (SVR) approach to TP. We can learn regression models $f_i : x \mapsto F(x, \mathcal{T}_{xx_i} \boldsymbol{q}_i)$ for the convergence costs F for each trajectory $\boldsymbol{q}_i \in D^y$ given some situation descriptor x. Then we can use these multiple models to find the index of the trajectory with lowest costs and the trajectory prediction model using data D' would be:

$$f(x) = \mathcal{T}_{xx_{\hat{i}}} \boldsymbol{q}_{\hat{i}} \ , \quad \hat{i} = \underset{i \in D}{\text{argmin}} \ f_i(x) \quad (21)$$

This method allows to use any regression method to predict the convergence costs when applying a trajectory from D in a given situation. With more complex models and enough training data we can learn complex functions mapping situation to cost of movement initialization for each individual movement in D. A drawback is that the set D becomes a fixed action set and the predicted trajectories will always come from it, so we can't generalize for motions outside of the set D. We also don't learn a general notion of situation similarity and can't interpret the features meaningfully with this prediction method.

6 Discussion

6.1 TP and Direct Policy Learning

Direct Policy Learning (DPL) is one of the fundamental approaches for imitation learning, see Argall et al (2009) for an overview. DPL tries to find a policy $\pi : s \mapsto a$ that maps state to action given observed state-action pairs (s, a). Given a parameterization of the policy, DPL is usually a supervised classification or regression problem. Usually the data comes from observation of an expert demonstration, and no assumptions of a cost function characterizing good motions is made.

In the motion planning framework we can get large amounts of demonstration data from simulation, and use it to learn motion policies that can generalize to various situations. We do not need to reproduce the demonstrated movements perfectly with TP, since we assume there is a cost function as in Equation (1) specifying what good motions are, and a planner will refine these motions subsequently after the initial initialization by minimizing the costs. The essence of TP is to find trajectories that can lead such a planner quickly to good local optima of the cost function landscape.

6.2 TP as Macro Action Policy

Another important question is why we chose to have whole trajectories as prediction output, macropolicy instead of micropolicy. A micropolicy would be in this case a mapping for every time step $\pi : x_t \to y_t$. Here y_t is the predicted movement command in some task space and x_t is the current situation descriptor, possibly changing at each time step. By iteratively predicting a

movement y_t and recalculating the situation descriptor x_t after executing the movement, one can build whole trajectories.

The mapping of a situation to such a small local movement step is a challenging machine learning problem, since we have to account for global paths and the locally shortest path to the target is a dead end if the robot is trapped. A possible approach to remedy this would be to build networks of states connected via local actions Stolle and Atkeson (2007). However, this can lead to jagged movements and fail to improve the planner behavior, as our results with RRT planners will show. Constructing such a network and searching for a global solution is also computationally expensive.

Using trajectories as macropolicies makes sense for our setup, since we use as data the local motion planner output: whole trajectories $\boldsymbol{q}$ and their costs.

7 Experiments

We examine several simulated task setups in which our robot, a Schunk LWA3 arm and a SDH hand, has to achieve a task by minimizing a cost function. For all scenario setups we examined, we generate random scenario instances (situations) by moving randomly objects around the workspace. Trajectory prediction learns from a set of demonstrated situations and movements and learns to generalize this behavior to new situations from the same generating distribution. For all tasks we planned kinematically with $T = 200$ time steps, which is reasonable time resolution for movements lasting a few seconds. For training the SVR approach to TP we used a polynomial kernel of degree 4 and penalty parameter $c = 1$. For training the similarity metrics and minimizing the loss from Equation (20) we used the Matlab optimization toolbox. The training time of both TP approaches was a few minutes only, negligible compared to the time for creating the datasets D'. For all planning algorithms and IK we used our own C++ implementation on a Pentium 2.4ghz computer. This section will proceed with a description of the three tasks we examined and the cost functions defining them. All the cost functions were defined so that a movement with cost less than 0.5 is good.

7.1 Reaching on Different Table Sides

7.1.1 Scenario Setup

In the first setup we examined, contains the robot arm which has to reach a target across a table of size (1.2,0.7,0.1) with the finger as endeffector, around an obstacle (the table) as seen in Figure 12. We controlled the 7 DoF of the arm, and the endeffector was defined as the tip of the hand. Different scenarios are generated by uniformly sampling the position of the table in an area of size (0.9, 02, 0.2), the target in area size (0.5, 0.2, 0.6), and the initial endeffector position in (0.3, 0.3, 0.9). Situations with initial collisions were not allowed. Too easy situations where the endeffector was closer than 30cm to the target were discarded in order to avoid trivial situations and to put a greater focus on more challenging scenarios, where the endeffector must move on the other side of the table to reach the target.

We used the standard cost function in equation (1) for reaching, penalizing collisions, keeping within joint limits and enforcing smoothness and precision at the endeffector position. We chose the term h to enforce a trajectory of short length with smooth transitions between the trajectory steps. We define h as

$$h(q_t, q_{t-1}) = \|q_t - q_{t-1}\|^2 \tag{22}$$

The cost term g in (1) is defined as

$$g(q_t) = g_{collision}(q_t) + g_{reach}(q_t) + g_{limit}(q_t) \tag{23}$$

where $g_{collision}$ penalizes collisions while executing the grasp movement. The value of this collision cost is the sum of the pairwise penetration depths c_i of colliding objects. Minimizing it moves the robot body parts away from obstacles.

$$g_{collision}(q_t) = 10^5 \sum_i c_i^2 \tag{24}$$

The task of reaching the target position with the endeffector is represented in g_{reach}. We want the target to be reached at the end of the movement, so we define this cost function to have a higher value for $t = T$:

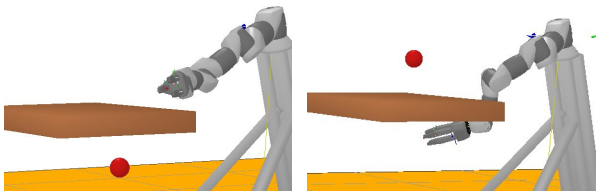
$$g_{reach}(q_t) = \begin{cases} 10^{-2} d^2 & t < T \\ 10^2 d^2 & t = T \end{cases} \tag{25}$$

where d is the Euclidean distance between the endeffector and the target.

The cost term g_{limit} puts limits on the joint angles:

$$g_{limit}(q_t) = 10^{-2} \sum_{i=1}^{n} \Theta(d_i - 0.1)^2 \tag{26}$$

where d_i is the distance of joint i from its limit (0 and 2 radians respectively), 0.1 is a margin, and Θ is the heavyside function.



(a) Situation x_a – move hand under the table. (b) Situation x_b – move hand over the table.

Fig. 2 Two situations; the goal is to reach the target.

7.1.2 Trajectory Prediction Setup

Since we have only one obstacle in this table reaching scenario, we used the geometric descriptor defined in Section 3.1, to see how well we can predict trajectories using high-dimensional geometric situation information. Concretely, the descriptor $x \in \mathbb{R}^{770}$ is defined as a 770-dimensional vector comprising all the information relevant for this setup. We have 11 objects for which we measure pairwise geometric information: 7 segments of the robot arm, the endeffector, the robot immobile platform (similar to the world frame), the largest obstacle object (a single table in our scenario) and the reach target location, shown in Figure 3(a). This makes 110 object pair combinations.

The first demonstration set D has 64 optimal situations and optimal movements. We also examined whether smaller subsets from D (created using K-means clustering and Euclidean distance on the task space trajectories y) used for the creation of D' as in Equation (14) can also work well to provide initialization options. Our results on the next figures show that as expected more trajectories d lead to better possible initializations. However, small numbers d provides already a variety of initial movements and allow good initialization with TP, so this can be tuned as necessary for different robot tasks with different computational costs.

To learn the similarity metric and predictor f we measured the costs F of these initial movements $\boldsymbol{q}_i$ in all 1000 situations $x_j \in D^x = D$, using $j = 20$ iterations and early stopping as defined in Equation (13).

To validate the results for the predictor f, we split the set D' by dividing D^x in 800 situations for training and 200 for testing the predictors. This way we can reason about generalization to new unseen situations of our predictors, or in other words transfer to new situations of motions evaluated on the train set situations.

The possible choices for prediction methods, including both trivial and trained TP predictors, are

- NNOpt staying for the nearest neighbor predictor from Section 5.2, with $\lambda = 0.0001$
- NNEuclid for nearest neighbour without training, with $w = 1$ the default Euclid metric
- SVR regression for method in Section 5.3
- *best* corresponding to a predictor always taking the trajectory from set D^y with smallest cost F
- *mean* for a predictor choosing a random trajectory

Figure 4(a) shows 3 different task spaces and their usefulness for initialization. The space Y_{target} represents movements relative to the target. The space Y_{obst} (with best performance in Figure 4(a))consists of endeffector coordinates relative to the largest obstacle, see figure 3(c). The joint space Q had poor performance, which confirms the hypothesis that joint space coordinates generalize poorly.

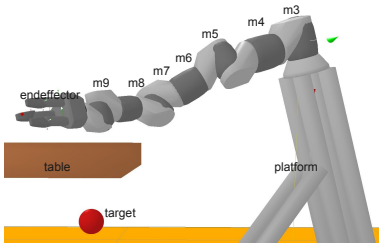
In Figure 4(b) we examine the performance of the different predictors f using data from the task space Y_{obst}. SVR and NNOpt have similar performance, and improve on both *mean* and NNEuclid. However, they are still away from the lower bound of performance *best*, which means that more complex models for similarity or regression can improve the performance further. The graphic also illustrates the trend that more motions in set D lead to better initializations. The regularization used for NNOpt also managed to compress the descriptor quite well: from 770 to 25 dimensions, as shown in Figure 3(b). The best features are the big table obstacle, the target, and the endeffector, which seems intuitively appealing interpretation of the reaching around table scenario.

We also tested varying the number of train situations of set D', as shown in Figure 4(c), and testing on the same 200 test situations. A difference between SVR and NNOpt is that NNOpt required significantly less training data for good performance. With as few as 25 situations on which all 64 movements are evaluated NNOpt can reach good prediction quality. In contrast, SVR needs at least 400 train situations to get good prediction quality on the validation set.

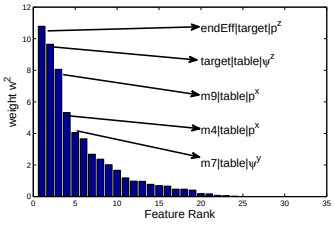
7.1.3 Planning Results

We present results for the average motion planning costs of the local optimizers and initializations as time progresses. The results presented are for 200 random test situations on which we already validated the predictors in the previous subsection. We evaluated 6 different motion generation methods by combining different initializations and planners. We tested three different initialization methods:

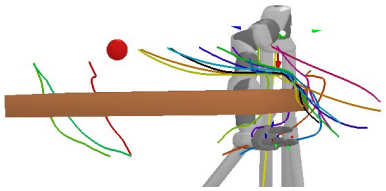
- LINEAR
- TP
- RRT



(a) The 11 landmarks used for the descriptor x: the centers of the 11 marked objects in the scene.

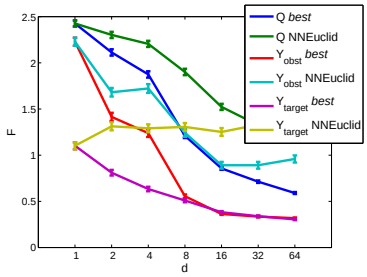


(b) The 25 nonzero features in the learned metric NNOpt, and the geometric information of the top 5.

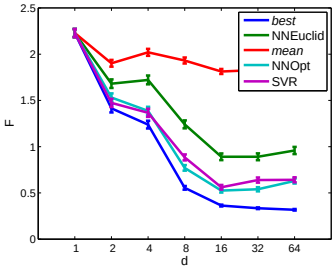


(c) Visualization of endeffector movement trajectories from D in space Y_{obst}, centered on table.

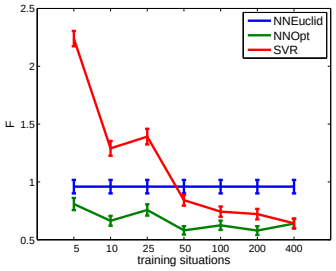
Fig. 3 Table reaching scenario: geometric landmarks, extracted features, and stored trajectory dataset



(a) The 3 trajectory task spaces compared: joint space Q is worst.



(b) Different prediction strategies for $f(x)$ in space Y_{obst}: number of movements d vs convergence costs F.



(c) Varying the number of situations used for training predictors $f(x)$ in space Y_{obst}.

Fig. 4 Table reaching scenario: convergence costs F averaged over 200 test situations and using d motions for initialization.

LINEAR is the default option, where the start and goal endeffector positions are connected with a straight line path, which is followed by the robot hand using IK for initialization. TP uses the *NNOpt* nearest neighbour predictor in the task space Y_{obst}. Both trajectory prediction and straight line initialization require an IK operator from the endeffector path to joint space. The time for the IK operator ϕ^{-1} was 0.07s. The TP prediction itself is practically instantaneous. RRT initialization uses our implementation of a standard algorithm for sampling collision free joint states. The RRT algorithm always works from scratch and does not require any training experience of the situations one can encounter. The creation of a RRT tree with 2000 nodes takes 8s, which is already a drawback for real time action and much slower than the other two initializations. However, we include RRT for performance comparison of the usefulness of such initial random collision free paths, ignoring this huge initialization time, assuming that some more efficient implementations of the RRT algorithm can do this faster.

The 3 initializations are combined with 2 different planner methods:

- iLQG
- AICO

For iLQG an initial trajectory $\tilde{\boldsymbol{q}}$ is already a part of the algorithm. For AICO we had to use $\tilde{\boldsymbol{q}}$ in a different way: only for the *first* iteration we use instead of $C(x, \boldsymbol{q})$ the cost $\tilde{C}(x, \boldsymbol{q}, \tilde{\boldsymbol{q}}) = C(x, \boldsymbol{q}) + \|\boldsymbol{q} - \tilde{\boldsymbol{q}}\|^2$. This forces the solution to be near $\tilde{\boldsymbol{q}}$ and changes the belief states of AICO respectively.

Both iLQG (Todorov and Li 2005) and AICO (Toussaint 2009) are local planners well suited for motion planning, as mentioned in the introduction. We set the iLQG convergence rate parameter $\epsilon = 0.8$; performance was robust with respect to different values of ϵ. For AICO we used instead of a fixed step parameter a second order Gauss Newton method to determine the step. One iteration of each of the planners took 0.07s, the bulk of which goes to collision detection and that is why the timings are similar for different planners.

We also tested direct gradient descent in joint space with the RPROP general optimization algorithm (Igel et al 2005), but its performance was an order of magnitude worse than the other 2 planners, so we did not add it to the final results.

The results in Figure 5 show the convergence behaviour of the planners for 7 seconds, and they allow us to make the following observations:

- TP is the best initialization for both AICO and iLQG, both speeding up convergence in the first

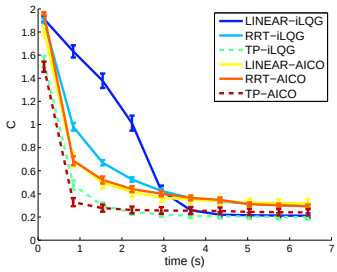


Fig. 5 Performance of different methods in table reaching scenario. The average cost C of the planners during their convergence is plotted versus time in seconds.

initializations, and allowing to reach solutions with lower costs overall. Sometimes a first feasible solution is reached in less than a second for TP-iLQG, in comparison to 3 seconds for LINEAR-iLQG.
- TP-AICO also benefits greatly from a TP initialization. Note that the data D' for prediction was gathered only with iLQG planner data, so our predictors could transfer successfully to a new planner.
- The RRT path initializations are unexpectedly poor choices for planner initialization: they start collision-free and with lower costs, but they are difficult for the planners to improve and after some planner iterations even LINEAR finds better overall solutions.
- AICO is potentially very sensitive to initialization: with improper initialization (from any of the 3 initialization methods we examined) it can converge to bad solutions, which are unlikely to be improved. iLQG is more robust in this sense: with more iterations bad solutions can still be improved.

7.2 Reaching in Cluttered Scene.

7.2.1 Scenario Setup

In the next setup, the table (from the previous experiment) is cluttered with 4 obstacles. Rectangles of various sizes and on random positions stand in the way of a target to be reached, as shown in Figure 6. We controlled the 7 DoF of the arm, and the endeffector was defined as the tip of the hand. The obstacle positions are randomly put over the table surface, and the target is put over the table to a place unoccupied by obstacles. We took the reaching cost defined in Section 7.1.1.

7.2.2 Trajectory Prediction Setup

For such cluttered situations we decided to test the sensor voxel descriptor $x \in \mathbb{R}^{413}$ from Section 3.2, since it is a compact way to represent the obstacle information. In the simulations we simulated an arm-mounted laser sensor delivering point cloud information to the scene, similar to Jetchev and Toussaint (2010).

For task space we examined again the 3 choices from the previous experiment: joint space Q, table relative coordinates Y_{obst} and target relative coordinates Y_{target}, shown in Figure 9(c). Y_{target} worked the best for this scenario.Some of the situations required complex avoidance paths, so the linear initialization failed often to find any solutions. Thus for the database D of optimal movements we had to use RRT initialization, otherwise the dataset D' was created identically as in Section 7.1.

In Figure 7(a) we compare the same 5 prediction methods defined in the previous section. For this task the SVR approach worked better than the nearest neighbour approaches. One possible explanation is that the Gaussian model for the similarity in equation (15) is not the optimal for such descriptors.

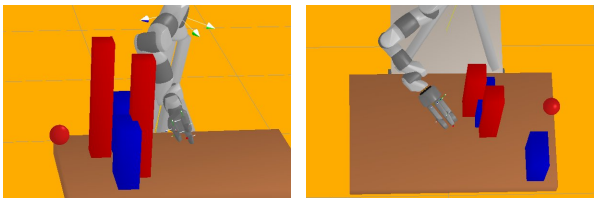
In Figure 7(b) we examine how many PCA components are necessary to create voxel descriptors good enough for predictive purposes. With 200 components (covering 99 % of the variance) the SVR regression achieves the best result. The NNOpt method can't handle well the voxel grid PCA components features and more components don't help.

7.2.3 Planning Results

The results presented are for 200 random test situations, different than the train situations. We tested the same 6 motion generation methods, but used instead of *NNOpt* the SVR approach for the trajectory prediction function in the task space Y_{target}. Each planner iteration and IK operation costs 0.15s. This is more than in the previous scenario due to more expensive collision check operations with more objects. This was the timing using the object models in the simulator. If we were to use the voxel representations obtained from analysis of point cloud data the timings would rise even more.

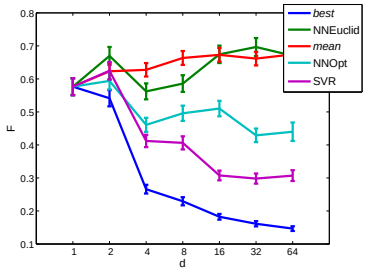
Figure 7(c) summarizes our planning experiments:

- for both AICO and iLQG planners, TP has lower costs than the RRT and LINEAR initializations.

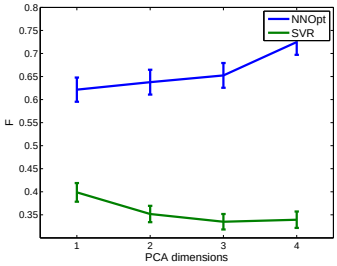


(a) (b)

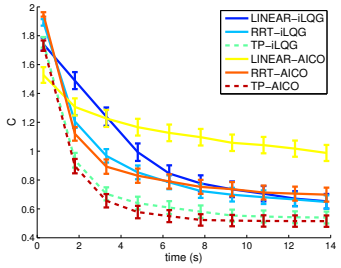
Fig. 6 Typical situations on cluttered tables with red point as target and table and 4 obstacles.



(a) Different prediction strategies for $f(x)$: number of movements d vs convergence costs F.



(b) Varying the number of PCA components used for $v = VP$ vs convergence costs F (with $d = 64$ motions).



(c) The average cost C of the planners during their convergence is plotted versus time in seconds.

Fig. 7 Costs in a cluttered table scenario and task space Y_{target}.

This is to be expected since LINEAR is often starting in collisions. RRT is collision free, but suffers from the random nature of the path construction, see Figure 9.

– LINEAR-AICO is prone to failure even after many iterations: the many obstacles make for a highly nonlinear cost surface with multiple local optima, and AICO gets stuck in suboptimal solutions.

We also tested a setup with 3 more obstacles, more difficult because obstacle avoidance paths become more complex. TP remained the fastest initialization even with this more cluttered setup, a transfer of useful behavior from the training setup with 4 blocks. This showed that the descriptors x and the predictor f can transfer knowledge to a more diverse set of scenarios without modification, since the occupancy of the workplace is represented well by the voxel descriptor x. On the other side, when considering the potential effect of adding even more objects in the scenario (e.g. more than 20), *RRT* has the best chance to solve such puzzles. The design of the scenario has big effect on performance.

In addition to simulation, we also did hardware tests as in Figure 8, and had robust performance in real scenes with different obstacles on tables.

7.3 Grasping a Cylinder

7.3.1 Scenario Setup

We also tested TP on tasks more complex than reaching. Grasping is one such task. The grasp setup we devised contains a long target cylinder of radius 5cm that has to be grasped by the robot, see Figure 10. For the random situations we translated the cylinder center in a rectangular area $(0.9, 0.5, 0.4)$ and rotated it around its radial axis by random angles in $(0, 2\pi)$. We also moved the hand at random starting position similar to the previous scenarios. We controlled both the arm and hand for this setup, resulting in a 14DoF joint space $q \in \mathbb{R}^{14}$.

Fig. 8 The Schunk robot arm, the SDH hand and an arm-mounted Hokuyo URG-04LX laser.

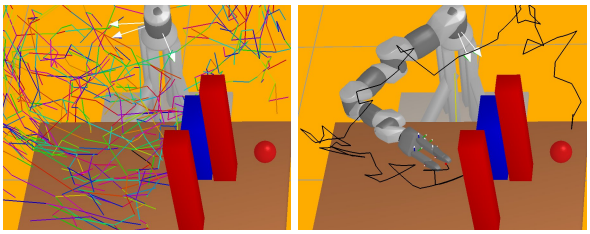
The cost function had the same smoothness term h, but a different term g:

$$g(q_t) = g_{collision}(q_t) + g_{limit}(q_t) + g_{surface}(q_t) \qquad (27)$$

where the collision and joint limit terms are the same as before. The term $g_{surface}$ measures the distance from the target surface to some markers on the robot body and forces the robot to move these markers on top of the surface. We defined 12 such markers, 3 on each of the 3 robot fingers, and 3 on the wrist. By taking a configuration of 3 markers near the surface of the fingers we force the robot to also align the fingers with the grasp target object, which leads to better grasps. The definition of $g_{surface}$ is:

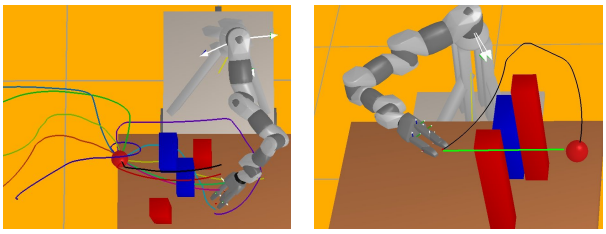
$$g_{surface}(q_t) = \begin{cases} 10^{-3} \sum_{i=1}^{18} \eta_i^2 & t < T \\ 10^{2} \sum_{i=1}^{18} \eta_i^2 & t = T \end{cases} \qquad (28)$$

where each η_i stays for distance to target cylinder surface of each of the 18 markers. This cost function is similar to the one used by Dragiev et al (2011).



(a) A RRT of random collision free samples accessible from the start position.

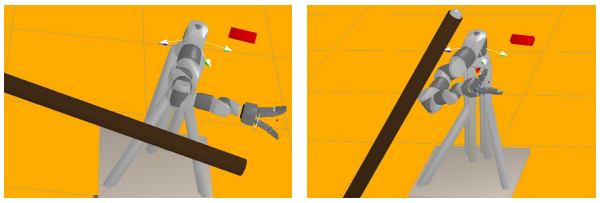
(b) The shortest path in this tree to the target is very inefficient if we want a smooth movement.



(c) Movement trajectories from D in space Y_{target}.

(d) A smooth movement from TP prediction (black). LINEAR (green) goes straight to the target and has high collision costs.

Fig. 9 A visualization of different initializations for a cluttered situation reaching task.



(a) (b)

Fig. 10 Typical situations in the grasping scenario: the cylinder is rotated and translated randomly.

The target can be grasped anywhere with this cost, but the challenge lies in positioning the robot fingers on the surface without colliding with it.

7.3.2 Trajectory Prediction Setup

We used here the geometric descriptor from Section 7.1.2, but with a slightly different object set: the 7 robot arm segments, the endeffector, the target cylinder center targetC, and a marker on top of the cylinder targetE. This results in 90 pairwise object distance descriptors and a situation descriptor $x \in \mathbb{R}^{630}$.

We examined 2 task spaces. First, the joint space $Q \in \mathbb{R}^{14}$. Second, $Y_{\text{qhand+target}} \in \mathbb{R}^{10}$ consisting of the 7 hand joints and the 3D relative position of the arm in the target frame. The sizes of datasets D and D' were as in the previous experiments, with the only different being that we needed $j = 40$ planner iterations

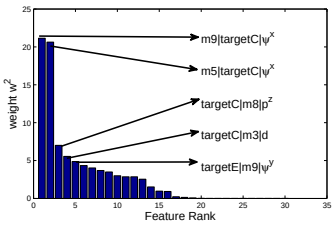


Fig. 11 Grasping task: the 17 nonzero features in the learned metric NNOpt, and the geometric information of the top 5 features.

to measure cost F, which made data gathering slower. $Y_{\text{qhand+target}}$ is better task space, see Figure 13(a): the relative positions of the hand in the target frame generalize well to target rotations and move the hand to positions which can be grasps near the cylinder surface, and the finger joint information moves the fingers in an appropriate pregrasp shape.

The regularization used for *NNOpt* also managed to compress the descriptor quite well: from 630 to 17 dimensions, as shown in Figure 11.

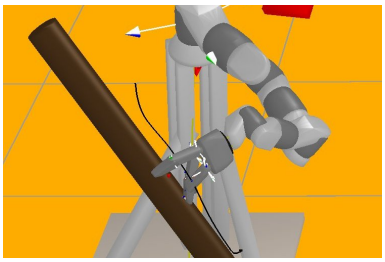
In Figure 13(b) we compare the 5 different predictors for good trajectory. The SVR predictor was the best, closely followed by NNOpt.

7.3.3 Planning Results

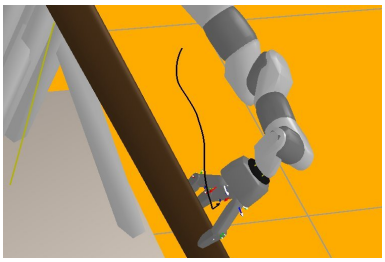
We tested 4 combinations of planner and initialization: AICO and iLQG combined with LINEAR initialization (with target the center of the cylinder) and TP initialization. We did not test RRT for grasping, since it would require major modifications to the default RRT algorithm.

In this scenario the time for a single planner iteration and IK transfer was 0.15s, and optimization to convergence required sometimes as much as 100 iterations and was with a high failure rate, so this problem has the potential to gain a lot from TP. Figure 13(c) shows our results for this more complex task:

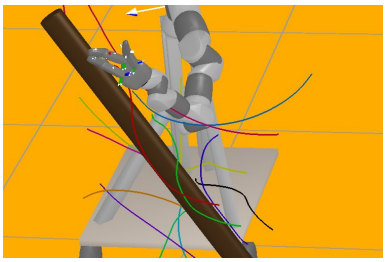
- the combination TP-AICO finds the best solutions overall: 2 seconds planning time with the TP initialization vs 14 seconds for LINEAR-AICO. TP-iLQG is similarly superior to the default LINEAR-iLQG.
- for the grasping task AICO is better than iLQG. A possible explanation can be the different task essence: for grasping the challenge is to coordinate multiple body parts to do a more complex movement, whereas for the previous two tasks the challenge is to avoid collisions. It seems that the inference algorithm of AICO can handle complex motions better than collision avoidance.



(a) A grasping movement: first approach (black line)

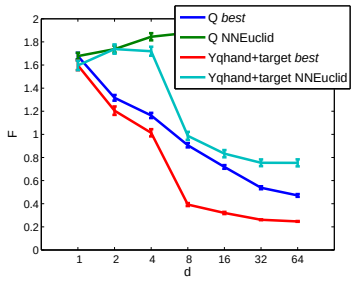


(b) Finally the fingers should close on the object surface

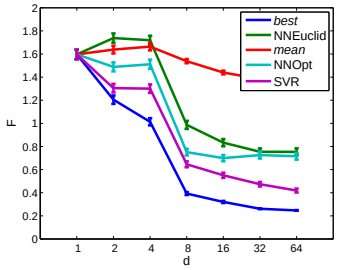


(c) Trajectories: arm and hand movements relative to the target frame. The arm movement is visualized.

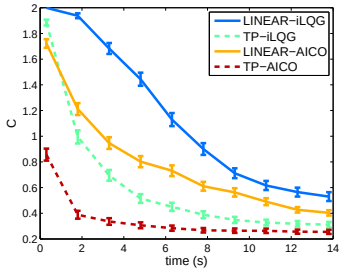
Fig. 12 An example grasping movement and the set of initializations visualized in space $Y_{\text{qhand+target}}$.



(a) The two trajectory task spaces Q and $Y_{\text{qhand+target}}$: number of movements d vs convergence costs F.



(b) Different prediction strategies for $f(x)$: number of movements d vs convergence costs F.



(c) The average cost C versus time in seconds for different planners.

Fig. 13 Costs of different methods in cylinder grasping scenario in the task space $Y_{\text{qhand+target}}$.

8 Conclusion

In this paper we proposed a novel algorithm to improve local motion planning methods. Trajectory prediction can exploit data from previous trajectory optimizations to predict reasonable trajectories in new situations. We proposed two key aspects to solve this problem: an appropriate situation descriptor and a task space transfer of previously optimized trajectories to new situations. Concerning the situation descriptor, we demonstrated that learning a (L_1-regularized) metric in a high-dimensional descriptor space significantly increases performance of the mapping. Interestingly, this means that we can extract features of a situation (e.g., choose from a multitude of possible coordinate systems) that generalize well w.r.t. trajectory prediction. The extracted features allow for an intuitive explanation of the crucial latent factors to choose one movement over another. The task space transfer – that is, first projecting an old trajectory to a task space and then projecting it back in the new situation, allows an adaptation to the new situation implicit in the inverse kinematics.

Speeding up local planners is crucial for fluid robot interaction with the world and humans. The TP framework for movement prediction is of great practical utility for many motion planning tasks, as shown by our experiments. A good initialization makes the planner converge faster. Additionally, the planners can converge to potentially better solutions which are not likely to be discovered by a naive initialization not using the experience of movement and situations incorporated by the trained predictors.

Future work will focus on making trajectory prediction less dependent on designer choices. In our current implementations, selecting the task space for motion transfer is important for the performance of the method. Developing data-driven methods for finding such task spaces using the demonstrated optimal motions will be a step further toward understanding the latent structure of motions.

Another possible direction is applying trajectory prediction to more complex and realistic scenarios, with sensor uncertainty and moving obstacles in the workspace under strict time constraints. Speeding up motion planning in such situations will be of even greater utility, especially if combined with parallel exploration of alternative predicted trajectories.

Acknowledgements This work was supported by the German Research Foundation (DFG), Emmy Noether fellowship TO 409/1-3.

References

Argall BD, Chernova S, Veloso MM, Browning B (2009) A survey of robot learning from demonstration. Robotics and Autonomous Systems 57(5):469 – 483

Atkeson CG (1993) Using local trajectory optimizers to speed up global optimization in dynamic programming. In: NIPS, pp 663–670

Bertram D, Kuffner J, Dillmann R, Asfour T (2006) An integrated approach to inverse kinematics and path planning for redundant manipulators. In: IEEE Int. Conf. on Robotics and Automation (ICRA), pp 1874–1879

Branicky M, Knepper R, Kuffner J (2008) Path and trajectory diversity: Theory and algorithms. In: IEEE Int. Conf. on Robotics and Automation (ICRA), pp 1359–1364

Calinon S, Billard A (2005) Recognition and reproduction of gestures using a probabilistic framework combining PCA, ICA and HMM. In: 22nd Int. Conf. on Machine Learning (ICML), pp 105–112

Dragiev S, Toussaint M, Gienger M (2011) Gaussian process implict surface for object estimation and grasping. In: IEEE Int. Conf. on Robotics and Automation (ICRA)

Dyer P, McReynolds SR (1970) The Computation and Theory of Optimal Control. Elsevier

Elfes A (1989) Using occupancy grids for mobile robot perception and navigation. Computer 22(6):46–57

Hiraki K, Sashima A, Phillips S (1998) From Egocentric to Allocentric Spatial Behavior: A Computational Model of Spatial Development. Adaptive Behavior 6(3-4):371–391

Igel C, Toussaint M, Weishui W (2005) Rprop using the natural gradient. Trends and Applications in Constructive Approximation International Series of Numerical Mathematics 151:259–272

Jetchev N, Toussaint M (2010) Trajectory prediction in cluttered voxel environments. In: Int. Conf. on Robotics and Automation (ICRA)

Kavraki LE, Latombe JC, Motwani R, Raghavan P (1995) Randomized query processing in robot path planning. In: Twenty-seventh annual ACM Symposium on Theory of Computing (STOC), pp 353–362

Konidaris G, Barto A (2006) Autonomous shaping: knowledge transfer in reinforcement learning. In: 23rd Int. Conf. on Machine Learning (ICML), pp 489–496

Martin S, Wright S, Sheppard J (2007) Offline and online evolutionary bi-directional RRT algorithms for efficient re-planning in dynamic environments. In: IEEE Int. Conf. on Automation Science and Engineering (CASE)., pp 1131–1136

Muehlig M, Gienger M, Steil JJ, Goerick C (2009) Automatic selection of task spaces for imitation learning. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pp 4996–5002

Nakhaei A, Lamiraux F (2008) Motion planning for humanoid robots in environments modeled by vision. In: 8th IEEE-RAS Int. Conf. on Humanoid Robots, pp 197–204

Peshkin L, de Jong ED (2002) Context-based policy search: Transfer of experience across problems. In: ICML-2002 Workshop on Development of Representations

Ratliff N, Zucker M, Bagnell A, Srinivasa S (2009) Chomp: Gradient optimization techniques for efficient motion planning. In: IEEE Int. Conf. on Robotics and Automation (ICRA)

Shon A, Storz J, Rao R (2007) Towards a real-time bayesian imitation system for a humanoid robot. In: IEEE Int. Conf. on Robotics and Automation (ICRA), pp 2847–2852

Stolle M, Atkeson C (2007) Transfer of policies based on trajectory libraries. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pp 2981–2986

Todorov E, Li W (2005) A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In: Proc. of the American Control Conference, vol 1, pp 300–306

Toussaint M (2009) Robot trajectory optimization using approximate inference. In: 26th Int. Conf. on Machine Learning (ICML), pp 1049–1056

Wagner T, Visser U, Herzog O (2004) Egocentric qualitative spatial knowledge representation for physical robots. Robotics and Autonomous Systems 49(1-2):25 – 42

Zacharias F, Borst C, Hirzinger G (2007) Capturing robot workspace structure: representing robot capabilities. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pp 3229–3236

Zhang J, Knoll A (1995) An enhanced optimization approach for generating smooth robot trajectories in the presence of obstacles. In: Proc. of the European Chinese Automation Conf., pp 263–268