Auton Robot (2013) 34:111–127
DOI 10.1007/s10514-012-9315-y

Fast motion planning from experience: trajectory prediction for speeding up movement generation

Nikolay Jetchev · Marc Toussaint

Received: 19 July 2011 / Accepted: 14 December 2012 / Published online: 12 January 2013

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. 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 tajectory pediction. The main idea is to use demonstrated optimal motions to quickly predict appropriate trajectories for novel situations. These can be used to initialize and thereby drastically speed-up subsequent optimization of robotic movements. Our approach has two essential ingredients. First, to generalize from previous situations to new ones we need a situation descriptor—we construct features for such descriptors and use a sparse regularized feature selection approach to improve generalization. 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. We present extensive results in simulation to illustrate the benefits of the new method, and demonstrate it also with real robot hardware. Our experiments in diverse tasks 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

N. Jetchev (✉) · M. Toussaint
Machine Learning and Robotics Lab, FU Berlin,
Arnimallee 7, 14195 Berlin, Germany
e-mail: nikolay.jetchev@fu-berlin.de

M. Toussaint
e-mail: marc.toussaint@fu-berlin.de

1 Introduction

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 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). The current journal article represents an expanded version of our previous work (Jetchev and Toussaint 2009, 2010; Jetchev 2012).

[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.

In Sect. 2 we will examine the basics of motion planning as optimization, and then present TP and how it is coupled with planning. In Sect. 3 we will discuss connections between TP and imitation learning. Finally, in the experimental Sect. 4 we will show our results for several simulated robot motion planning scenarios.

These are the main contributions:

- 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 and generalization to new situations,
- the notion of IK transfer in task space, allowing robust adaptation of the predicted movements to different situations,
- quantitative results in three different motion planning tasks.

We finish the current introduction with an overview of related methods.

1.1 Related motion and trajectory generation methods

Our method TP builds on classical motion generation methods from the rich toolbox of robotics. Here is a small overview of these 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 that minimize this cost. Popular approaches use spline-based representation and gradient descent (Zhang and Knoll 1995), covariant gradient descent (Ratliff et al. 2009), Differential Dynamic Programming (Dyer and McReynolds 1970; Atkeson 1993), the related iterated Linear Quadratic Gaussian (iLQG) (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 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 (RRTs) and other sampling methods

Another approach for finding good movement trajectories is sampling to find obstacle free paths in the configuration and workspace of the robot, i.e. finding an appropriate initialization of the movement plan. Popular methods for planning feasible paths without collisions are RRTs (LaValle 2006) which use random sampling 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

Our approach TP works by fusing the classical planning methods with machine learning. The idea to utilize data for motion generation has a long history. Here we give a short overview of related works, and in Sect. 3 we will discuss further how TP is different than some of these methods.

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.

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. Bruce and Veloso (2002) work on fast replanning with RRTs in fixed environments. Martin et al. (2007), Zucker et al. (2008) 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 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.

Jiang and Kallmann (2007) exploit information from previous RRT paths by using attractors to bias sampling. A simple notion of situation similarity is used, but it uses only a small set of specified features; there is no notion of feature selection as in our work.

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, this method is adapted specifically to the locomotion navigation domain by combining local step planning with global graph-based search. It does not learn data-driven situation feature representations and does not attempt to generalize to different tasks, unlike our approach TP.

The field of imitation learning (Argall et al. 2009) encompasses many approaches that use demonstrated motions to learn behaviors, usually 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, 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. Ude et al. (2010) work on making such imitation more adaptable. In contrast, our method TP is designed to work for motion planning tasks with a known cost function, a scenario different than imitation learning.

Kober et al (2010) worked on learning parameters of the planner to improve planning with experience and maximize expected rewards. Lampariello et al. (2011) takes a similar approach for real-time robot motion planning. Both these works differ from our TP method in that they make strong assumptions about a small set of relevant features. They have analogies to our approach: a predicted trajectory is a kind of "meta-parameter" for the motion planner. However, the trajectories need IK transfer to be adapted between situations, unlike planner parameters that can be directly reused.

2 The TP method

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 trajectory can be calculated by a planner module minimizing a cost function. One can think of the behavior of the 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, then formalizing TP, and finally discussing the components of the algorithm, namely the situation descriptor, the task space transfer and the motion-to-situation mapping.

2.1 Robot motion planning: a basic model

Let us describe the robot configuration as $q \in \mathbb{R}^N$, the joint posture vector. A motion trajectory $q(t),\ t \in \mathbb{R}$ describes the joints in time. For simplicity we assume a time discretization $(t_0, \ldots, t_T)$ for T time steps is given. We write a trajectory formally as

$$\boldsymbol{q} = (q_0, \ldots.., q_T) = (q\,(t_0), \ldots, q\,(t_T)) \in \mathbb{R}^{N\times T}. \quad (1)$$

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 Sect. 2.3), 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

$$C(x, \boldsymbol{q}) = \sum_{t=1}^{T} g_t\,(q_t) + h_t\,(q_t, q_{t-1})\,. \quad (2)$$

This cost characterizes the quality of the joint trajectory in the given situation and task constraints. We will specify such a cost function explicitly in our Sect. 4. 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,

$$x \mapsto \boldsymbol{q}^* = \underset{\boldsymbol{q}}{\operatorname{argmin}}\, C(x, \boldsymbol{q}). \quad (3)$$

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 nonlinear 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, see Ratliff et al. (2009). RRT can be used for finding good initial paths, but has the drawback of a higher computational burden: construction of tree with collision free steps.

2.2 TP overview

In this section we first define the TP 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 (3) from a dataset of previously optimized trajectories. The dataset D comprises pairs of situations and optimized trajectories,

$$D = \left\{ \left(x_i, \boldsymbol{q}_i\right)_{i=1}^{d} \right\}, \qquad \boldsymbol{q}_i \approx \underset{\boldsymbol{q}}{\operatorname{argmin}}\, C\left(x_i, \boldsymbol{q}\right). \tag{4}$$

Figure 1 and Algorithm 1 summarize the idea of TP and the steps involved in training and testing it. The full sequence involved in predicting a trajectory for a new situation is

$$x \to \hat{i} \to \mathcal{T}_{x x_{\hat{i}}} \boldsymbol{q}_{\hat{i}} \to \mathcal{O}_x \mathcal{T}_{x_{\hat{i}} x} \boldsymbol{q}_{\hat{i}} = \boldsymbol{q}^*. \tag{5}$$

TP takes as input an appropriately represented situation descriptor x, see Sect. 2.3. 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 $x_{\hat{i}}$ to x, described in Sect. 2.4. We can view the subsequence

$$f : x \to \mathcal{T}_{x_{\hat{i}} x} \boldsymbol{q}_{\hat{i}}, \tag{6}$$

as the policy mapping situation to motion, and will explain it in Sect. 2.5. 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 dataset 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.

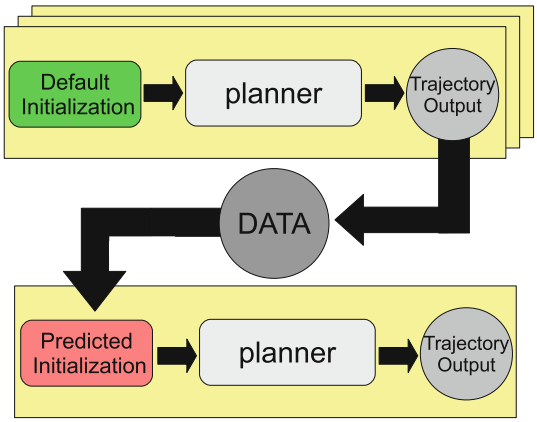


Fig. 1 A *diagram* illustrating TP: we gather trajectory data from multiple runs of the motion planners; this data can then be used to predict a smarter initialization speeding-up planner performance in a novel situation

Algorithm 1 Trajectory Prediction (TP): steps required for training on data and generating new motion.

1: define a task cost C
2: define a situation descriptor x as in Section 2.3
3: define a task space mapping ϕ as in Section 2.4
4: supply an IK operator ϕ^{-1} and planner operator $\mathcal{O}$
5: create data D and D' as in Section 2.5.1
6: train a mapping f as in Section 2.5
7: for a new situation x generate motion as in Equation (5)

2.3 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 in this section two different approaches for modeling x, appropriate for scenarios with different assumptions.

2.3.1 General geometric descriptor

Our first approach is to define a very high-dimensional and redundant situation descriptor which includes distances and relative positions wrt many different frames of reference. Training the predictive function then includes selecting the relevant features. Assume we have a set of b different three-dimensional (3D) objects (i.e. landmarks) in the scene which might be relevant for motion generation $A = (a_1, \ldots, a_b)$ 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 $i = (i_1, i_2) \in \{1, \ldots, \hat{b}\}$ 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 7D vector $\phi_i = (p_i, d_i, \psi_i)$. The final descriptor x comprises all these local pairwise vectors

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

Given such a descriptor we can use a feature selection technique to infer from the data which of these dimensions are best for TP in new situations. In the experimental Sect. 4 we will show how extracting a sparse representation from this redundant description provides an interesting explanation of the important factors in a situation.

2.3.2 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 TP. Since modeling each of these as an object with coordinates in the descriptor x is impractical, we can use instead voxel information for 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:

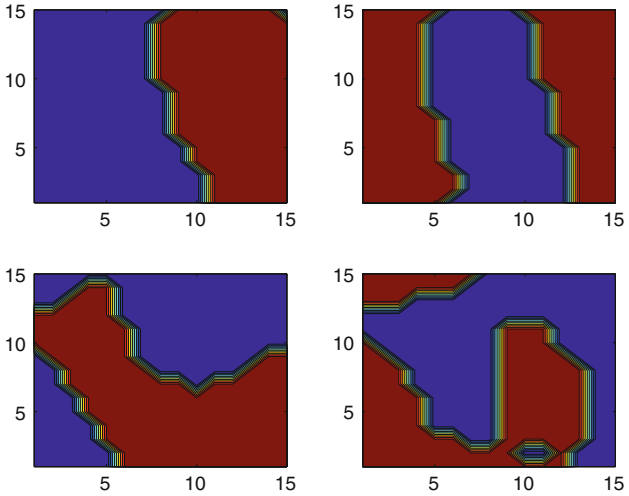


Fig. 2 The first four PCA components, visualized in the *square* plane of size 15×15 voxels from a slice 7 cm above the *center* of the grid. *Red areas* are more likely to be occupied, and *blue areas* are probably free (Color figure online)

(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.

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 7 cm. 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 $15^3 = 3,375$ 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 $\hat{V} \in \mathbb{R}^{200}$. In Fig. 2 we show the column vectors of the PCA projection matrix, 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 the compressed representation is good for distinguishing between different motions types.

The final situation descriptor is then

$$x = \left\{ d,\ \hat{V}_1,\ \hat{V}_2 \right\} \in \mathbb{R}^{413}. \tag{8}$$

The entries $\hat{V}_1,\ \hat{V}_2$ 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.

2.4 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.

2.4.1 Motion representation for output trajectory task space

As we mentioned in Sect. 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 good planner initialization, similar 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.

2.4.2 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 Eq. (5). 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 Eq. (2)) to transfer motions and adapt to new situations.

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

$$C^{IK}\left(x,\ q,\ q_{t-1},\ q'_t\right) = \underbrace{g(q)}_{\text{task cost}} + \underbrace{h\left(q,\ q_{t-1}\right)}_{\text{small step}} \tag{9}$$

$$+ \underbrace{\left\|\phi_x(q) - \phi_{x'}\left(q'_t\right)\right\|^2}_{\text{follow } \phi_{x'}(\boldsymbol{q}')}, \tag{10}$$

where q'_t is a step from the motion $\boldsymbol{q}'$ that is being transferred. The terms of C^{IK} are chosen so that making steps with low C^{IK} fulfill different criteria important for motion adaptation. The terms h and g influence the motion steps to have a low cost with respect to the terms of the (task-specific) cost function from Eq. (2). Usually the term $h(q,\ q_{t-1}) = \|q - q_{t-1}\|^2$ is used to force smooth, energy efficient motions. The term $g(q)$ stays for additional criteria for good motions, e.g. avoiding collisions. The term $\|\phi_x(q) - \phi_{x'}(q'_t)\|^2$ is for following the task space trajectory.

We generate a motion for each time slice $t = 1, \ldots, T$ by using IK control

$$q_t = \phi_x^{-1}\left(q_{t-1},\ q'_t\right) \approx \underset{q}{\operatorname{argmin}}\ C^{IK}\left(x,\ q,\ q_{t-1},\ q'_t\right). \tag{11}$$

The mapping $\phi_x^{-1} : (\boldsymbol{y},\ q_0) \mapsto \boldsymbol{q}$ projects the whole task space trajectory back to a joint space trajectory in situation x, applying the IK operator ϕ_x^{-1} to minimize next-step cost C^{IK} iteratively for each time step t, starting from q_0.

The transfer operator is then the function composition

$$\mathcal{T}_{xx'}\boldsymbol{q}' = \phi_x^{-1} \circ \phi_{x'}(\boldsymbol{q}'). \tag{12}$$

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.

2.5 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.

2.5.1 Gathering data demonstrating behavior

The first step toward learning f is the recording the output of a planner of optimal trajectories in different random situations, as formalized in Eq. (4). Afterwards we can gather data D' for the quality of the initialization using the new actions in different situations. What we are really interested in is how much additional refinement a certain initialization would need from the planner. We measure this "refinement cost" as

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

Here $\mathcal{O}_x^j \boldsymbol{q}$ is the trajectory vector found by the optimizer after j iterations, starting from initialization $\boldsymbol{q}$. Such a definition of F is a heuristic to quantify the effect of initialization on convergence speed looking only at a limited number 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

$$\begin{aligned} D' &= \left\{\left(x_j,\, x_i,\, F\left(x_j,\, \mathcal{T}_{x_j x_i} \boldsymbol{q}_i\right)\right)\right\}, \\ x_j &\in D^x, \quad \left(x_i,\, \boldsymbol{q}_i\right) \in D. \end{aligned} \tag{14}$$

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 Sect. 4 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.

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 as input to IK transfer.

Once we have gathered data in the set D' as defined in Eq. (14), we can use it to learn the TP mapping from Eq. (6). 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.

2.5.2 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 TP quality.

We define the situation similarity function as

$$\begin{aligned} k\left(x,\, x_i\right) &= \exp\left\{-\frac{1}{2}\left(x - x_i\right)^T W \left(x - x_i\right)\right\}, \\ W &= \operatorname{diag}\left(w_1^2, \ldots, w_s^2\right). \end{aligned} \tag{15}$$

The nearest neighbor predictor (NNOpt) f for x is

$$f(x) = \mathcal{T}_{x x_{\hat{i}}} \boldsymbol{q}_{\hat{i}}, \qquad \hat{i} = \operatorname*{argmax}_{i \in D} k\left(x_i,\, x\right). \tag{16}$$

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

$$P\left(f(x) = \mathcal{T}_{x_i x} \boldsymbol{q}_i\right) = \frac{1}{Z} k\left(x,\, x_i\right), \tag{17}$$

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

We can define the expectation over the planner costs in situation x when initializing with Eq. (17) as

$$\mathrm{E}\{F(x,\, f(x))\} = \sum_{i \in D} P\left(f(x) = \mathcal{T}_{x x_i} \boldsymbol{q}_i\right) F\left(x,\, \mathcal{T}_{x x_i} \boldsymbol{q}_i\right). \tag{18}$$

Our goal is to find a similarity metric with low expected motion planning costs. For this purpose we define the training loss function L using the cross initialization data D'

$$L(w;\, D') = \frac{1}{|D^x|} \sum_{x \in D^x} \mathrm{E}\{F(x,\, f(x))\} + \lambda |w|_1. \tag{19}$$

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 Sect. 4.

2.5.3 TP via cost prediction

As an alternative to the above prediction scheme and for empirical evaluation we also test a cost prediction 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. The TP model using data D' is

$$f(x) = \mathcal{T}_{xx_{\hat{i}}}\boldsymbol{q}_{\hat{i}}\,, \qquad \hat{i} = \operatorname*{argmin}_{i \in D} f_i(x). \tag{20}$$

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.

3 Discussion of TP

In this section we will discuss two important aspects of TP: why it is different than imitation learning of the observed optimal motions, and why we predict *whole* trajectories at once.

3.1 Difference to direct policy learning

The direct policy learning (DPL) approach to learning from demonstration (Pomerleau 1991) has some analogies with our TP method, but also numerous differences. 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's (teacher's) behavior. No assumption of a cost function characterizing good motions is made, and the prediction is to be made only with the criteria to reproduce the expert motion accurately. This is *mimicry*: attempting to repeat an action without understanding its goal.

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 Eq. (2) and a planner. The essence of TP is to find trajectories that can lead such a planner quickly to good local optima of the cost function landscape. For this we need to predict a trajectory just once, in the starting situation, and rely that the structure in this trajectory will improve the planner performance. This is *goal emulation*: attempting to attain the goal of observed actions without caring whether the action is duplicated accurately, see Call and Carpenter (2002). In the case of TP the goal is to obtain low cost motion planner output, which means usually going to a desired target configuration with low costs on the way. The predicted trajectory initialization is the action coming from our prediction policy, but it is transformed by additional planner iterations. The final planner output that the robot will follow in the world (the action of TP in another sense) can be quite different from the initial trajectory. This is acceptable, since only the low cost of the final trajectory matters for motion planning purposes. Penalizing deviation from the initialization is not a criteria in the given cost function.

3.2 TP as a macro-action policy

Another important question is why we chose to have *whole* trajectories as prediction output, a *macro-action* (McGovern and Sutton 1998), instead of the micropolicy used by DPL, namely a mapping for every time step $\pi : x_t \rightarrow y_t$. 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 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. Such reactive policy can not anticipate and adapt for a longer time horizon. 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. Another reason why micropolicies are more difficult to learn is that we would need to make accurate predictions for the whole state space, whereas for TP we need to learn mappings just for the smaller subset of starting situations in our dataset.

Using trajectories as macropolicies makes sense for our setup for several reasons. First, we use as input data the local motion planner output: whole trajectories $\boldsymbol{q}$ of length T steps and their costs. Second, we need to learn a predictive mapping saying which of these trajectories are good initialization for a given situation Third, we need to output a whole trajectory $\boldsymbol{q}$ of length T to initialize a motion planner

4 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. TP 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 cost prediction approach to TP we used a support vector regression (SVR) with a polynomial kernel of degree 4. For training the similarity metrics and minimizing the loss from Eq. (19) 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 2.4 GHz 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.

4.1 Reaching on different table sides

The first task we examined was reaching in the presence of an obstacle. Though basic, it represents a good benchmark for the quality of motion planners.

4.1.1 Scenario setup

The reaching setup contains the robot arm which has to reach a target with the finger as endeffector, around an obstacle (a table), as shown in Fig. 3a. 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, the target, and the initial endeffector position. Situations with initial collisions were not allowed. Too easy situations where the endeffector was closer than 30 cm 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 standard terms in the cost function Eq. (2) 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

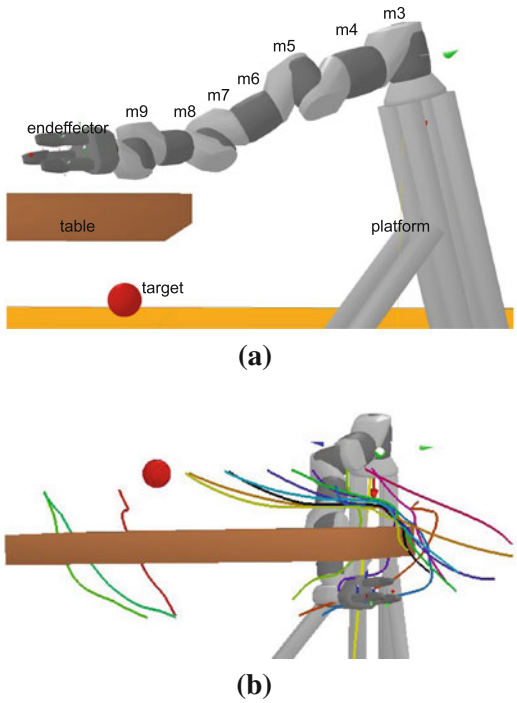


Fig. 3 Table reaching scenario: geometric landmarks and stored trajectory dataset. **a** The 11 landmarks used for the descriptor x: the centers of the 11 marked objects in the scene. **b** Visualization of endeffector movement trajectories from D in space Y_{obst}, centered on table

$$h\left(q_t,\, q_{t-1}\right) = \left\| q_t - q_{t-1} \right\|^2 . \tag{21}$$

The cost term g is defined as

$$g\left(q_t\right) = g_{collision}\left(q_t\right) + g_{reach}\left(q_t\right) + g_{limit}\left(q_t\right) . \tag{22}$$

The term $g_{collision}$ is equal to the collision cost, namely the sum of the pairwise penetration depths c_i of colliding objects. Minimizing it moves the robot body parts away from obstacles.

$$g_{collision}\left(q_t\right) = 10^5 \sum_i c_i^2 . \tag{23}$$

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 with a higher weight for the final step $t = T$

$$g_{reach}\left(q_t\right) = \begin{cases} 10^{-2} d^2 & t < T \\ 10^2 d^2 & t = T \end{cases} . \tag{24}$$

In the definition d is the Euclidean distance between the endeffector and the target.

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

$$g_{limit}(q_t) = 10^{-2} \sum_{i=1}^{n} \Theta(d_i - 0.1)^2 . \quad (25)$$

In this equation d_i is the angular distance (in radians) of joint i from its limit, 0.1 is a margin, and Θ is the heavyside function.

4.1.2 TP setup

Since we have only one obstacle in this table reaching scenario, we used the geometric descriptor defined in Sect. 2.3.1, 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 770D vector comprising all the information relevant for this setup. We have 11 objects for which we measure pairwise geometric information: seven 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 Fig. 3a. This makes 110 object pair combinations.

The first demonstration set D has 64 optimal situations and optimal movements. We also examined using smaller subsets from D (created with K-means clustering) for the creation of D' as in Eq. (14). Our results on Fig. 5a, b 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 1,000 situations $x_j \in D^x = D$, using $j = 20$ iterations and early stopping as defined in Eq. (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.

We inspected five different prediction methods, including both trivial and trained TP predictors. NNOpt is our name for the nearest neighbor predictor from Sect. 2.5.2, with $\lambda = 0.0001$. NNEuclid stays for nearest neighbour prediction without training, i.e. using $w = 1$ the default Euclid metric. Third, we denote by SVR the cost regression approach from Sect. 2.5.3. *best* stays for a predictor always taking the trajectory from set D^y with smallest cost F. Finally, we write *mean* for a predictor choosing a random trajectory.

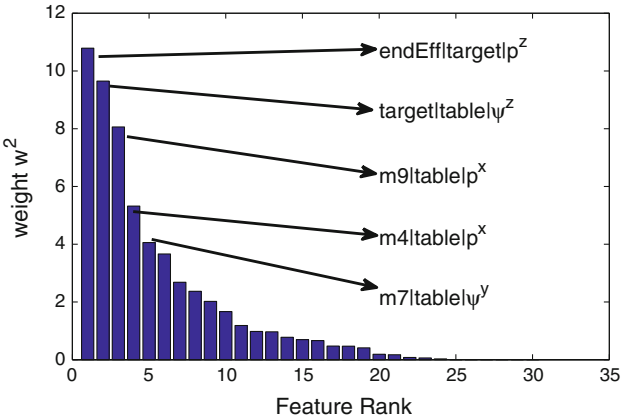


Fig. 4 The 25 nonzero features in the learned metric NNOpt, and the geometric information of the top 5

Figure 5a shows three 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 Fig. 5a)consists of endeffector coordinates relative to the largest obstacle, see Fig. 3b. The joint space Q had poor performance, which confirms the hypothesis that joint space coordinates generalize poorly.

In Fig. 5b 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 Fig. 4. 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 Fig. 5c, 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.

4.1.3 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.

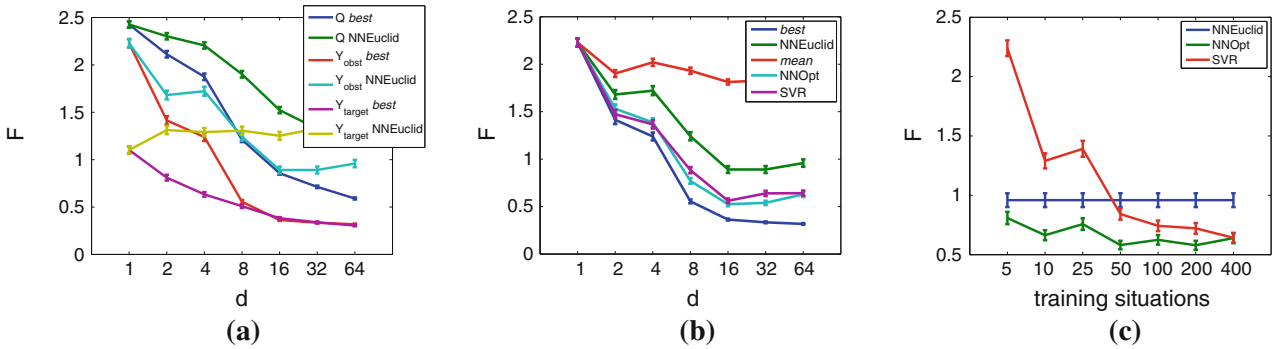


Fig. 5 Table reaching scenario: convergence costs F averaged over 200 test situations and using d motions for initialization. **a** The three trajectory task spaces compared: joint space Q is worst. **b** Different prediction strategies for $f(x)$ in space Y_{obst} : number of movements d versus convergence costs F. **c** Varying the number of situations used for training predictors $f(x)$ in space Y_{obst}

We tested three different initialization methods: LINEAR, TP and RRT. 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 in the task space Y_{obst}. Both TP and straight line initialization require an IK operator from the endeffector path to joint space. The time for the IK operator ϕ^{-1} was 0.07 s. The TP prediction itself is practically instantaneous. RRT initialization uses our implementation of a standard algorithm for sampling collision free joint states. The creation of a RRT tree with 2,000 nodes takes 8 s, 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 paths, ignoring this huge initialization time, assuming that some more efficient implementations of the RRT algorithm can be faster.

The three initializations are combined with two different planner methods: iLQG and AICO. For iLQG an initial trajectory $\tilde{\boldsymbol{q}}$ as input is expected by 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 0.07 s, the bulk of which goes to collision detection and that is why the timings are similar for different planners. We also tried direct optimization in joint space, but the performance was an order of magnitude worse than the other two planners, so we did not add it to the final results.

We evaluated six different motion generation methods by combining different initializations and planners, and for our results we name them as "INITIALIZATION-PLANNER", e.g. TP–iLQG uses TP for initialization followed by iLQG for optimization, hile TP–AICO follows TP with AICO planning. These two methods starting with TP are used to validate the performance of our novel approach.

The results in Fig. 6 show the convergence behavior of the planners for 7 s (roughly 100 planner iterations), and they allow us to make the following observations. First, TP is the best initialization for both planners AICO and iLQG. TP speeds up convergence in the first initializations, and also allows 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 s for LINEAR–iLQG. The reason for this superb performance is that going around the table is a non-local behaviour. It puts the robot endeffector first away from the target, but the longer path allows a collision free reaching motion finally. Greedily going to the target leads LINEAR to collisions, but anticipating and predicting a detour trajectory (as TP does, see Fig. 7b) works well. Second, 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. Third, 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 (random paths are not smooth) and after some planner iterations even LINEAR finds better overall solutions. Fourth, AICO is potentially very sensitive to initialization: with improper initialization (from any of the three 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.

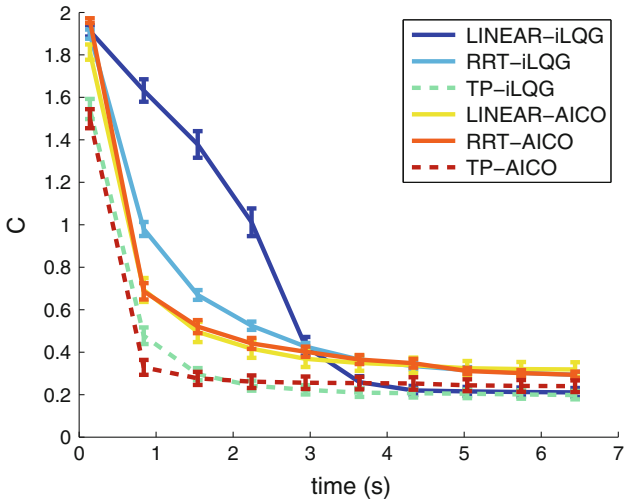
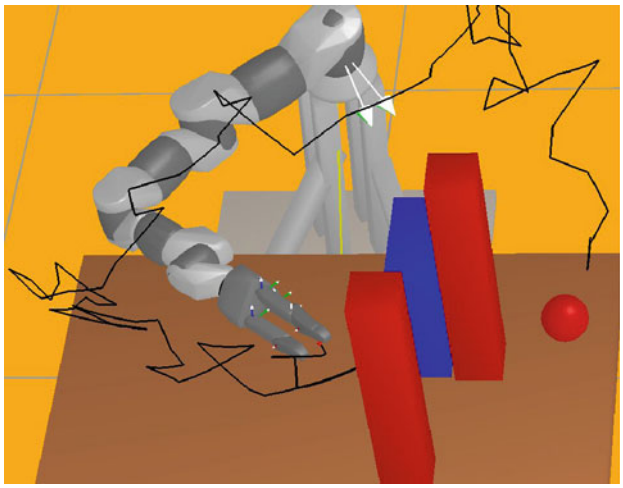
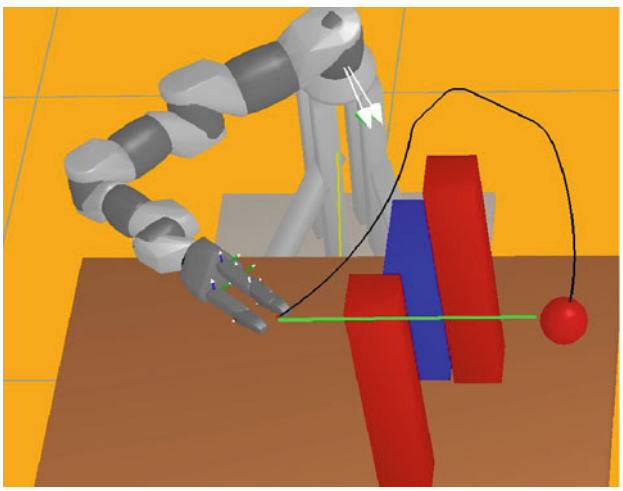


Fig. 6 Performance of different methods in table reaching scenario. The average cost C of the planners during their convergence is plotted versus time in seconds



(a)



(b)

Fig. 7 A visualization of different initializations for the cluttered situation reaching task. **a** The shortest path in a RRT tree to the target is very inefficient as input to a planner because the jagged path is difficult to smooth and optimize into an energy-efficient trajectory. **b** A smooth movement from TP prediction (*black*). LINEAR (*green*) goes straight to the target and has high collision costs (Color figure online)

4.2 Reaching in cluttered scene

Once we tested performance in scenes with a single obstacle, we proceed to test the performance of TP in scenes with multiple obstacles that represent a greater challenge.

4.2.1 Scenario setup

In the next setup, the table (from the previous experiment) is cluttered with four obstacles. Rectangles of various sizes and on random positions stand in the way of a target to be reached, as shown in Fig. 7. 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 Sect. 4.1.1.

4.2.2 TP setup

For such cluttered situations we decided to test the sensor voxel descriptor $x \in \mathbb{R}^{413}$ from Sect. 2.3.2, 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). 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 Sect. 4.1. The datasets we used for TP were of the same size as for the previous scenario. We used the task space Y_{target} for transfer.

In Fig. 8a 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.

4.2.3 Planning results

The results presented are for 200 random test situations, different than the train situations. We tested the same six motion generation methods, but used instead of NNOpt the SVR approach for the TP function in the task space Y_{target}. Each planner iteration and IK operation costs 0.15 s. This is more than in the previous scenario due to more expensive

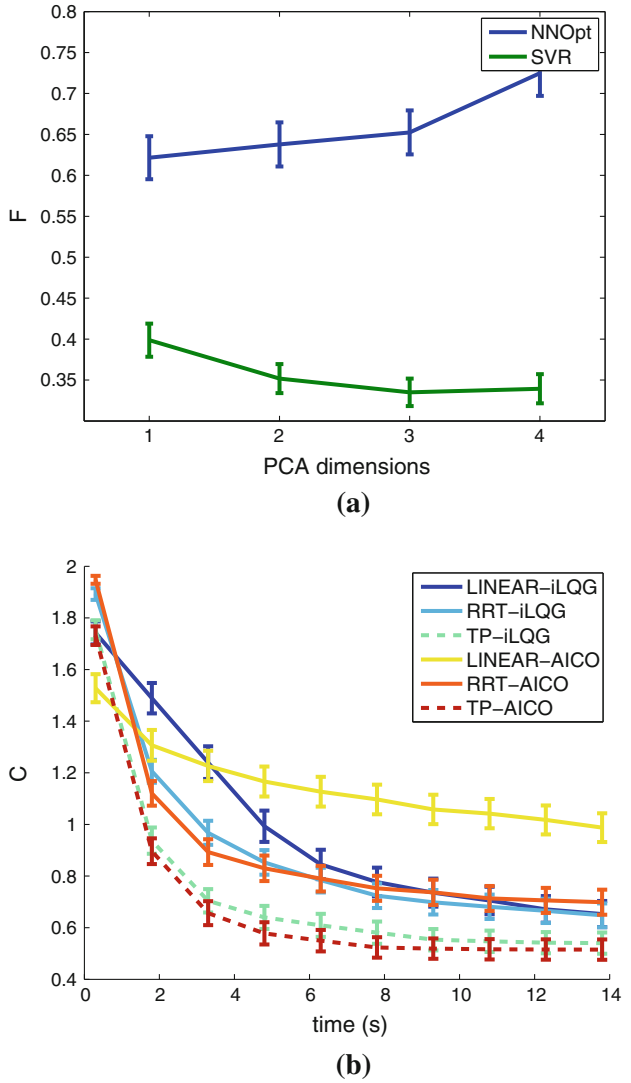


Fig. 8 Costs in a cluttered table scenario. **a** Varying the number of PCA components used for $v = VP$ versus convergence costs F (with $d = 64$ motions). **b** The average cost C of different initializations and planners is plotted versus time in seconds

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 8b summarizes our planning experiments, and we make two observations. First, for both AICO and iLQG planners, TP has lower costs than the RRT and LINEAR initializations. This is to be expected since LINEAR is often starting in collisions, while TP handles the obstacles better, see Fig. 7b. RRT is collision free, but suffers from the random nature of the path construction, see Fig. 7a. Second, 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.

Fig. 9 The Schunk robot arm, the SDH hand and an arm-mounted Hokuyo URG-04LX laser

We also tested a setup with three 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 four 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.

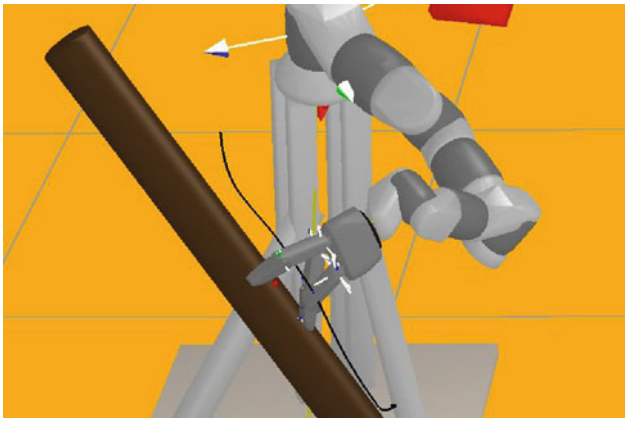
In addition to simulation, we also did hardware tests as in Fig. 9, and had robust performance in real scenes with different obstacles on tables.

4.3 Grasping a cylinder

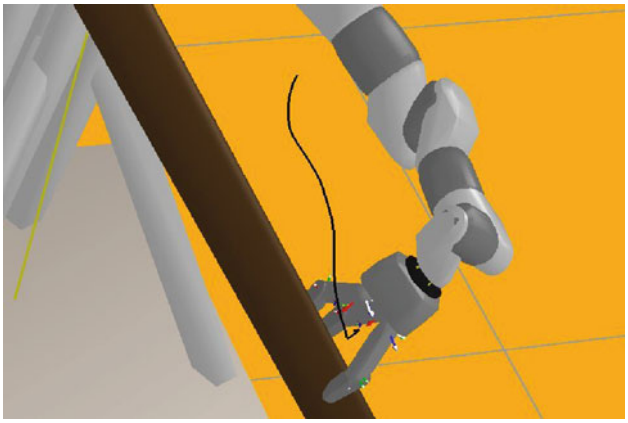
We also tested TP on tasks more complex than reaching. Grasping is one such task that requires motion of multiple body parts (an arm and hand with fingers) in a coordinated manner.

4.3.1 Scenario setup

The grasp setup contains a long target cylinder of radius 5 cm that has to be grasped by the robot, see Fig. 10. For the random situations we translated the cylinder center and rotated it around its radial axis. We also moved the hand at random starting position similar to the previous scenarios. We



(a)



(b)

Fig. 10 An example grasping movement. **a** A grasping movement: first approach (*black line*). **b** Finally the fingers should close on the object surface

controlled both the arm and hand for this setup, resulting in a 14 DoF joint space $q \in \mathbb{R}^{14}$. The cost function has the same smoothness term h, but a term g defined as

$$g(q_t) = g_{collision}(q_t) + g_{limit}(q_t) + g_{surface}(q_t). \quad (26)$$

Here the collision and joint limit terms are the same as in Eq. (22). The new 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 three 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:

$$g_{surface}(q_t) = \begin{cases} 10^{-3} \sum_{i=1}^{18} \eta_i^2 & t < T \\ 10^{2} \sum_{i=1}^{18} \eta_i^2 & t = T \end{cases}, \quad (27)$$

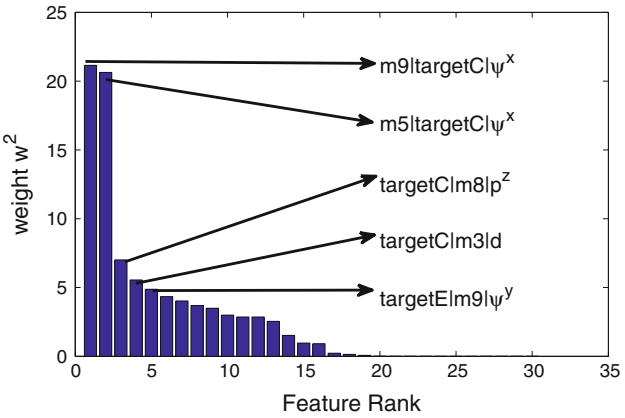


Fig. 11 Grasping task: the 17 nonzero features in the learned metric NNOpt, and the geometric information of the top 5

where each η_i stays for distance to target cylinder surface of each of the 18 markers.

4.3.2 TP setup

We used here the geometric descriptor from Sect. 4.1.2, but with a slightly different object set: the seven 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 two task spaces. First, the joint space $Q \in \mathbb{R}^{14}$. Second, $Y_{qhand+target} \in \mathbb{R}^{10}$ consisting of the seven 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 to measure cost F, which made data gathering slower. $Y_{qhand+target}$ is better task space, see Fig. 12a: 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 Fig. 11.

4.3.3 Planning results

We tested four 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 0.15 s, and optimization to convergence required sometimes

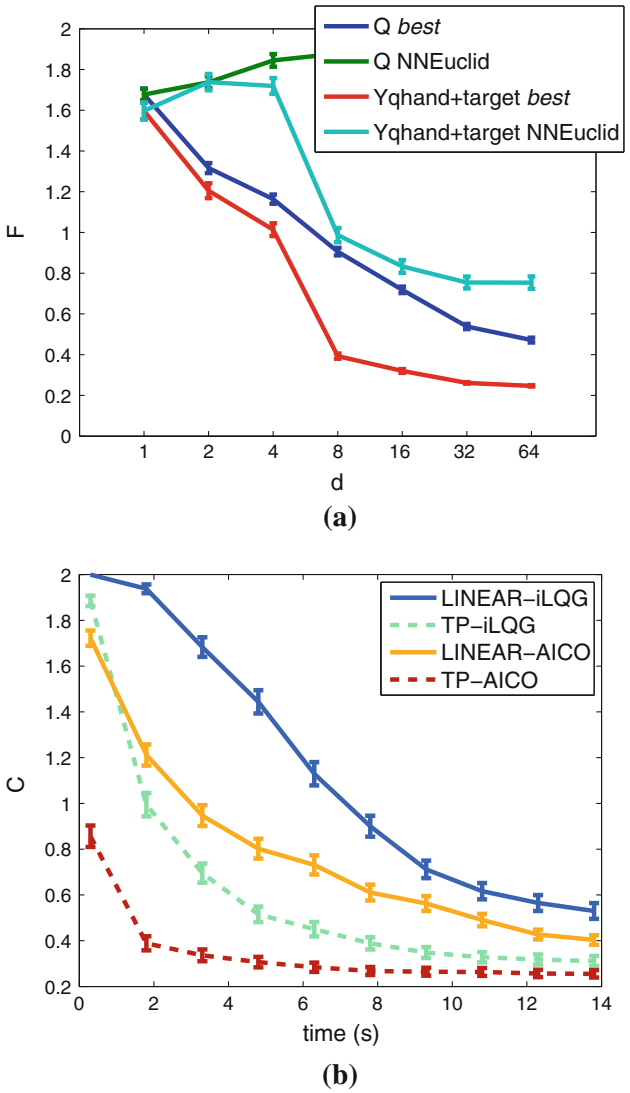


Fig. 12 Costs of different methods in cylinder grasping scenario in the task space $Y_{qhand+target}$. **a** The two trajectory task spaces Q and $Y_{qhand+target}$: number of movements d versus convergence costs F. **b** The average cost C versus time in seconds for different initializations and planners

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 12b shows our results for this more complex task. The combination TP–AICO finds the best solutions overall: 2 s planning time with the TP initialization versus 14 s for LINEAR–AICO. TP–iLQG is similarly superior to the default LINEAR–iLQG. A possible reason is that the grasp database D contains structure which is useful to be reused. Predicting a starting trajectory with correct finger closing behaviour and a hand pose approaching the target cylinder from a convenient angle helps the planner for the most challenging grasping aspects, namely positioning the robot fingers on the target surface without colliding with it.

For the grasping task TP–AICO is better than TP–iLQG, and also LINEAR–AICO is better than LINEAR–iLQG. Our explanation is that for grasping the challenge is to coordinate multiple body parts to do a more complex movement, whereas for the previous two tasks the challenge was mainly to avoid collisions. It seems that the inference algorithm of AICO can handle complex motions better than collision avoidance.

5 Conclusion

In this paper we proposed a novel algorithm to improve local motion planning methods. TP 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 wrt TP. 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. Using TP for movement prediction is beneficial 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.

In our current implementations, selecting the task space for motion transfer is important for the performance of TP. 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 TP 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 can be useful, especially if combined with parallel exploration of alternative predicted trajectories.

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

References

Argall, B. D., Chernova, S., Veloso, M. M., & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, *57*(5), 469–483.

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

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

Bruce, J., & Veloso, M. (2002). Real-time randomized path planning for robot navigation. In *International Conference on Intelligent Robots and Systems* (*IROS*), Switzerland.

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

Call, J., & Carpenter, M. (2002). Three sources of information in social learning. In K. Dautenhahn & C. L. Nehaniv (Eds.), *Imitation in animals and artifacts* (pp. 211–228). Cambridge, MA: MIT Press.

Dyer, P., & McReynolds, S. R. (1970). *The computation and theory of optimal control*. New York: 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.

Jetchev, N. (2012). Learning representations from motion trajectories: Analysis and applications to robot planning and control. PhD Thesis, FU Berlin. Retrieved August 1, 2012 from http://www.diss.fu-berlin.de/diss/receive/FUDISS_thesis_000000037417.

Jetchev, N., & Toussaint, M. (2009). Trajectory prediction: Learning to map situations to robot trajectories. In *26th International Conference on Machine Learning* (*ICML*) (pp. 449–456).

Jetchev, N., & Toussaint, M. (2010). Trajectory prediction in cluttered voxel environments. In *International Conference on Robotics and Automation* (*ICRA*) (pp. 2523–2528).

Jiang, X., & Kallmann, M. (2007). Learning humanoid reaching tasks in dynamic environments. In *International Conference on Intelligent Robots and Systems* (*IROS*) (pp. 1148–1153).

Kober, J., Oztop, E., & Peters, J. (2010). Reinforcement learning to adjust robot movements to new situations. In *Robotics*: *Science and Systems*.

Konidaris, G., & Barto, A. (2006). Autonomous shaping: Knowledge transfer in reinforcement learning. In: *23rd International Conference on Machine Learning* (*ICML*) (pp. 489–496).

Lampariello, R., Nguyen-Tuong, D., Castellini, C., Hirzinger, G., & Peters, J. (2011). Trajectory planning for optimal robot catching in real-time. In *IEEE International Conference on Robotics and Automation* (*ICRA*) (pp. 3719–3726).

LaValle, S. M. (2006). *Planning algorithms*. Cambridge: Cambridge University Press. Retrieved January 5, 2012 from http://planning.cs.uiuc.edu/.

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

McGovern, A., & Sutton, R. S. (1998). Macro-actions in reinforcement learning: An empirical analysis. Technical Report 98–70. Amherst, MA: University of Massachusetts.

Muehlig, M., Gienger, M., Steil, J. J., & Goerick, C. (2009). Automatic selection of task spaces for imitation learning. In *International Conference 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 International Conference on Humanoid Robots* (pp. 197–204).

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

Pomerleau, D. A. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, *3*, 88–97.

Ratliff, N., Zucker, M., Bagnell, A., & Srinivasa, S. (2009). Chomp: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference 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 International Conference on Robotics and Automation* (*ICRA*) (pp. 2847–2852).

Stolle, M., & Atkeson, C. (2007). Transfer of policies based on trajectory libraries. In *International Conference 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 *Proceedings of the American Control Conference* (Vol. 1, pp. 300–306).

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

Ude, A., Gams, A., Asfour, T., & Morimoto, J. (2010). Task-specific generalization of discrete and periodic dynamic movement primitives. *IEEE Transactions on Robotics*, *26*(5), 800–815.

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 *International Conference 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: *Proceedings of the European Chinese Automation Conference* (pp. 263–268).

Zucker, M., Kuffner, J., & Bagnell, J. A. D. (2008). Adaptive workspace biasing for sampling based planners. In *IEEE International Conference on Robotics and Automation* (*ICRA*).

Nikolay Jetchev is currently a postdoctoral researcher at the group of Prof. Toussaint. He finished in 2012 his PhD studies at the the Machine Learning and Robotics Lab in FU Berlin, under the supervision of Prof. Toussaint. He got his diploma in mathematics from TU Darmstadt in 2007. His main interests are representations for motions and situations, imitation learning, and inverse reinforcement learning.

Prof. Marc Toussaint is heading the Machine Learning and Robotics Lab at FU Berlin since 2010. Before he initiated this group at TU Berlin and spend his postdoc at University of Edinburgh. His main interest is in learning, planning and inference methods in MDPs, POMDPs and multi-agent problems as well as their application in robotics.