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

Discovering Relevant Task Spaces Using Inverse Feedback Control

Nikolay Jetchev, Marc Toussaint

Received: date / Accepted: date

Abstract

Learning complex skills by repeating and generalizing expert behavior is a fundamental problem in robotics. However, the usual approaches do not answer the question of what are appropriate representations to generate motion for a specific task. Since it is time-consuming for a human expert to manually design the motion control representation for a task, we propose to uncover such structure from data – observed motion trajectories. Inspired by Inverse Optimal Control, we present a novel method to learn a latent value function, imitate and generalize demonstrated behavior, and discover a task relevant motion representation. We test our method, called Task Space Retrieval Using Inverse Feedback Control (TRIC), on several challenging high-dimensional tasks. TRIC learns the important control dimensions for the tasks from a few example movements and is able to robustly generalize to new situations.

1 Introduction: Open Questions in Imitation Learning

A common approach and an important tool for training robots in complex tasks (Argall et al. 2009) is learning from demonstration: given examples of correct motions, learn a policy mapping state to action consistent with the training data. Using machine learning techniques to find structure and learn policies from demonstrations of desired behavior is often much more efficient than hand-crafting robot motion controllers.

Nikolay Jetchev is with FU Berlin, Marc Toussaint is with Uni Stuttgart
E-mail: nikolay.jetchev@fu-berlin.de, marc.toussaint@informatik.uni-stuttgart.de

However, the question of *"What to imitate?"*, i.e. which aspects of the observed motions should be duplicated, is not answered in general. Extracting task spaces from data is particularly relevant in the context of articulated robots. In order to generate complex movements for a certain task, a controller typically goes to a desired configuration in some movement representation. There can be many important features of motions, e.g. collision avoidance, hand positioning, finger alignment, etc. Which features are suitable and how they are weighted depends on the task at hand, and it is often not practical to have a human expert select manually the best features given a task. The challenge in imitation learning thus becomes to retrieve the latent movement representation which can then be used in a motion controller. As an example consider a robot grasping an object: from observing successful motions the robot can learn that some finger configurations relative to the object surface result in good grasps. This observation can lead to specialized representations useful for a grasping motion controller, much better than representation in the direct robot joint configuration space.

The question of a suitable motion representation for a given task is closely related to the issue of the underlying generative policy of a motion good for a certain task. Suppose we observe a set of motion trajectories and have no additional information about the goals and structure of the motions. We can ask the question *"Why were the motions performed in this way?"*. The implicit assumption we make is that the teacher that showed us these demonstrations is rational and the demonstrations were optimal with respect to some criteria defined in an appropriate feature subspace. Both the optimality criteria and the relevant motion features are not directly recorded in the observations. Both can be consid-

ered latent structure in the data and can be potentially learned. *Task Space Retrieval Using Inverse Feedback Control* (TRIC) addresses the above issues by discovering from data a sparse discriminative value function using the most relevant motion features and using them to generate motions.

We mention briefly few of the advantages of TRIC. First, it can handle example demonstrations in high dimensional spaces and select the important features for movement. Second, it can generalize well to situations and constraints unseen during demonstrations (e.g. grasping objects on different positions and collision avoidance with new obstacles), because the learned value function defines a task manifold, a whole set of desired states. Without explicit knowledge of relevant task dimensions and goals, the controller using the learned value function can effectively generalize motion from few training examples to new situations.

This section will proceed with a brief overview of related work. Afterwards in Section 2 we give more detailed background on related methods we compare with or build on. In Section 3 we will present our algorithm, give details of the loss function used to train it and how the discovered task structure is used for motion control. In Section 4 we will give a more detailed discussion and analysis of our algorithm. We will test the effectiveness of our method in the experimental Section 5. Finally, Section 6 concludes the article and gives some ideas for future work.

The current article enhances upon our previous work (Jetchev and Toussaint 2011; Jetchev 2012) in several ways. First, we have a new discussion Section 4 with insights about the way TRIC relates to other imitation learning methods. Second, in Sections 5.4 and 5.5 we give a detailed analysis of the properties of the learned value function and how it relates to the task spaces relevant for imitation. Third, in Section 5.3 we include a new experimental setup with another challenging task - manipulation of an articulated object.

1.1 Related Work in Imitation Learning

Imitation learning via Direct policy learning (DPL) (Pomerleau 1991) is a popular approach for learning from demonstration. In essence DPL takes demonstration data in the form of motion trajectories and uses supervised learning to estimate a controller that would reproduce the trajectories. Bain and Sammut (1996) call this method behavioral cloning and describe methods to use decision trees to extract automatically reactive agents from logged data of skilled human operators. In general, this approach works well when the exact motion reproduction is desired (gestures, point-to-point motions), but it can have difficulties to generalize and modify the behaviors in new situations.

Another approach to DPL (Howard et al. 2009) addresses a challenging setup for imitation: there are hidden constraints influencing the task, and all observations are subject to them. Their work tries to generalize to different constraints, possibly unseen during training, by uncovering a latent policy independent of the constraints affecting the training data.

Dynamic Movement Primitives (DMP) (Schaal et al. 2003) is an advanced method that learns motion controllers defined by a parametrized dynamic system. It has inspired many works of research in imitation learning. Ude et al. (2010) address the issue of imitating tasks and generalizing between different situations. Given a situation to achieve a task, the system queries its trajectory experience and weights the demonstrations by their similarity to the current situation. Khansari-Zadeh and Billard (2010) present a method for imitating motions by learning an underlying dynamic system, similar to DMP. The method can adapt to perturbations in the data and deal with multidimensional data.

However, all of the above works related to DMP still require to pre-specify a relevant task space in which to construct dynamical motion systems. They do not address the case when there are potentially thousands of task spaces and it is not clear which are relevant.

Inverse Optimal Control (IOC) , also known as Inverse Planning or Inverse Reinforcement Learning, provides a general framework to retrieve latent objectives (rewards or costs) in observed behavior. IOC takes demonstrations in some state space, and learns a state cost function that gives rise to a policy consistent with this data. For example, Ratliff et al. (2006) take as input examples of good driving trajectories and learn which terrain features are good for driving and which are to be avoided. Ratliff et al. (2009) describes a combination of IOC and behaviour cloning. It builds on the strengths of both methods: the long-horizon planning abilities of IOC and the high dimensional action selection capabilities of behavioural cloning. Our method will have a similar framework.

1.2 Previous Work in Recovering Task Spaces

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. In the human motion experiments of Berniker and

Kording (2008) the space of joint angles Q is called intrinsic coordinate system, and a relative position space is called extrinsic. Muehlig et al. (2009); Billard et al. (2004) examine different task spaces for robot motions and select the best ones for reproducing different tasks. However, both examine only a small pool of possible task spaces, whereas our method can find rich motion representations from high dimensional task spaces.

The question of suitable representations for state and action spaces has received some interest in the context of reinforcement learning. Nouri and Littman (2010) deal with dimension reduction in continuous model-based reinforcement learning, coupling feature selection with efficient exploration in order to find the relevant dimensions of the environment.

There are approaches to motion feature selection that look at data variance in an unsupervised way, see Jenkins and Matarić (2004); Calinon and Billard (2007). While this allows to find compact representations that compress the motion data, it is not at all guaranteed that these will be also the features useful for motion generation and control. In contrast to such methods uncoupled from the motion generation policy, our approach TRIC jointly addresses the problems of finding relevant features of the motion, learning the behavior to imitate, and retrieving a (latent in the demonstrations) value function leading to such behavior.

2 Background

In this section we will first briefly describe the notation we use for articulated robot motion, and then give details for three methods that will be relevant in understanding how TRIC works in the later sections.

2.1 Articulated Robot Motion

A robot consists of a set of joint axes connecting rigid body links. We denote by $q = \{q_1, q_2, ...q_n\} \in \mathbb{R}^n$ the vector of the robot joint angles. This is also known as the robot configuration space, and changing the values of these angles leads to the robot moving. A motion trajectory $\{q(t) : t \in [0, T]\}$ with time t describes the joint angles as they change in time. For simplicity we assume that a time discretization $(t_0, ..., t_T)$ for T time steps is given.[1] We write a trajectory formally as

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

A kinematic function ϕ uses the robot geometry and the current joint angles q to find the value of some motion feature $\phi : q \mapsto \phi(q) \in \mathbb{R}^d$. The features $\phi(q)$ are non-linear in q and can be computed from the robot kinematics. We would call motion features also *task spaces*, because they provide different representations to describe robot motions good for different tasks. A simple example for a motion feature $\phi(q)$ can be the 3D coordinates of the robot endeffector in the world frame, or in some other reference frame. The matrix of partial derivatives of a kinematic map is called the Jacobian matrix $\mathcal{J}(q) = \dfrac{\partial \phi(q)}{\partial q} \in \mathbb{R}^{d \times n}$. If $d = 1$ we will call $\mathcal{J}$ the gradient instead of the Jacobian.

[1] To simplify even more we use $t_0 = 1$ and $t_T = T$.

2.2 Direct Policy Learning

Given a robot trajectory $\boldsymbol{q}$, a standard approach to imitation learning would be to extract data from the trajectory of the form $\{x_t, u_t\}_{t=1}^T$, where x_t represents the robot state at time t and u_t is the control signal. E.g. x_t can be just q_t and u_t the rate of change $\dot{x_t}$. DPL would learn a policy $\pi : x_t \mapsto u_t$ from these observations. Different assumptions can be made for the choice of x,u and π (Calinon and Billard 2007), with refinements like data transformations and active learning. Given a parametrization of the policy, DPL essentially corresponds to a regression problem, e.g. with loss:

$$E_{\text{DPL}} = \sum_{t=1}^{T} \|\pi(x_t) - u_t\|^2 \ , \tag{2}$$

where $\|.\|^2$ denotes the squared L_2 norm. Minimizing E_{dpl} finds a policy close (in the least-squares sense) to the demonstrations. The above loss can be easily extended to multiple demonstration trajectories by summing over them.

Howard et al. (2009) introduces an interesting alternative loss for DPL:

$$E_{\text{inc}} = \sum_{t=1}^{T} (\|u_t\| - \pi(x_t)^T u_t / \|u_t\|)^2 \ . \tag{3}$$

This loss penalizes the discrepancy between the projection of the policy $\pi(x_t)$ on u_t and the true control u_t. It allows searching for an underlying unconstrained policy projected into the constraint nullspace. In some problem domains – specific demonstration setups where the observed motion was subject to varying unseen constraints and the learning task is to find the underlying unconstrained policy – this loss leads to better behavior than the standard least squares loss. For instance, if $u_t \propto \pi(x_t)$ then $E_{\text{inc}} = 0$ ensures that the controlled dynamical system generates approximately the same paths as the demonstrations but with different velocity profiles.

When the state and control spaces are high dimensional DPL has a disadvantage: generalization to new test data is an issue and would essentially require the training data to cover all possible situations. TRIC improves generalization by extracting the relevant task features from data and by finding an underlying structure of the demonstrated trajectories.

2.3 Inverse Reinforcement Learning

Reinforcement Learning (RL) and planning within a MDP framework (Puterman 1994) generate motions maximizing some reward (negative cost). Learning requires constant feedback in the form of rewards for the states and action of the agent. In some domains it is much easier to learn a mapping from state to reward than to learn a mapping from state to action, because the latter is a more complex and higher dimensional problem, especially when considering actions in high dimensional continuous spaces such as robot control. A simple reward function can lead to complex optimal policies and states with high reward create a task manifold, a whole space of desired robot positions good for the task, which allows to have more flexible motion controllers.

However, in many tasks the reward is not analytically defined and there is no way to access it from the environment accurately. It is then up to the human expert to design a reward leading the robot to the desired behavior. Inverse Optimal Control (IOC) learns a reward function only on the basis of data.

In the following we sketch how a variant of IOC, Maximum Margin Planning (Ratliff et al. 2006), works. Let's assume that a policy π gives rise to expected feature counts $\mu(\pi)$ of feature vectors ϕ over state-action pairs.The reward is a linear function in the features parametrized by w, and the parameters should be learned such that the behavior demonstrated by the expert π^* has higher expected reward (negative costs) than any other policy. Subject to these constraints, learning the globally optimal parameters w is achieved by minimizing the training loss

$$\min \ |\, w \,|^2 \tag{4}$$

$$s.t. \ \forall \pi \ \ w^T \mu(\pi^*) > w^T \mu(\pi) + \mathcal{L}(\pi^*, \pi) \ . \tag{5}$$

The term $w^T \mu(\pi)$ defines an expected total reward. In order to calculate $\mu(\pi)$, some IOC methods (Ratliff et al. 2006) require to forward simulate the states and actions that result following a policy π. While for some domains this simulation can be done efficiently, in general robot systems can be high-dimensional and require complicated physical simulations, which can be a costly operation.

In the equation above the scalable margin $\mathcal{L}$ penalizes those policies that deviate more from the optimal behavior of π^*. In order to avoid adding *every* policy π as a constraint, efficient methods are required to find the π that violates the constraints the most and add it as new constraint. Once the reward model is learned, another module is required to generate motions maximizing the reward, e.g. Ratliff et al. (2006) uses an A^* planner to find a path to a target with minimal costs.

2.4 Discriminative Learning

Discriminative learning provides a common framework for many learning problems, including structured output regression. Popular approaches include large margin models (Tsochantaridis et al. 2005) and energy based models using neural networks (LeCun et al. 2006). Data is given in the form of pairs of input and output values $\{x_i, y_i\}$. As in standard discriminative approaches (e.g., structured output learning), the energy or cost $f(x_i, y_i; w)$ provides a discriminative function such that the true output should get the lowest energy from the model f:

$$y_i = \operatorname*{argmin}_{y \in \mathcal{Y}} f(x_i, y) \ .$$

Training a parameter vector w of the model f is done by minimizing a loss over the dataset. The loss should have the property that f is penalized whenever the true answer y_i has higher energy than $\tilde{y}_i$, the false answer with lowest energy which is at least distance r away:

$$\tilde{y}_i = \operatorname*{argmin}_{y \in \mathcal{Y} : \|y - y_i\| > r} f(x_i, y) \ . \tag{6}$$

Finding the most offending answer $\tilde{y}_i$ is very often a complicated inference problem in itself.

In this article will be using the log loss for training TRIC:

$$L_{log}(x_i, y_i) = \log(1 + e^{f(x_i, y_i) - f(x_i, \tilde{y}_i)}) \ , \tag{7}$$

This is a soft form of $L_{hinge}(x_i, y_i) = \max(0, m + f(x_i, y_i) - f(x_i, \tilde{y}_i))$ with infinite margin (LeCun et al. 2006).

In the next Section 3 we will show in more detail our algorithm TRIC and its relation to discriminative learning for the purpose of learning a value function from motion demonstrations.

3 The TRIC Algorithm

This section will proceed to explain the TRIC method in detail. First we describe our motion model, that is,

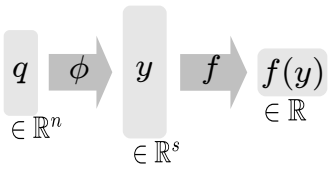


Fig. 1 Motion representation scheme: the joints q are mapped by ϕ to a high-dimensional feature vector y, the task space. which is used to find the value function $f(y)$. In general y has many more dimensions than q; f should be sparse in y.

how the value function f implies the movement. Second, we state the desired properties of the value function f and define the training loss for learning f from demonstrations.

3.1 Motion Model and Controller

TRIC is designed for generating motion trajectories for articulated robot models. We assume that given a joint configuration q we can compute a high-dimensional feature vector $\phi(q) \in \mathbb{R}^s$, which we will describe concretely in Section 5. It is the objective of learning to select relevant features to describe the value function f of a motion. This value function is defined in task space ϕ of the robot, so that TRIC can use the richness of the representation $\phi(q)$ to learn a value function model and select features, as shown in Figure 1.

We assumed that the teacher was rational with respect to some hidden criteria. Thus, if we suppose that f is trained as a value function consistent with the observations, the robot can generate motion with a method commonly used in robotics: motion rate control. More precisely, the robot motion is generated by imposing a smooth decreasing "motion" on f which is translated back to joint angle motions using Inverse Kinematics (IK), see Craig (1989); Siciliano and Khatib (2008).

More formally, motion rate control can be defined as computing a new robot pose given the old pose q_t in each time slice t by minimizing the next step cost C^{MR}:

$$q_{t+1} = \underset{q}{\operatorname{argmin}}\, C^{\mathrm{MR}}(q, q_t) \ . \tag{8}$$

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

$$C^{\mathrm{MR}}(q, q_t) = \underbrace{\varrho \| f \circ \phi(q) - f \circ \phi(q_t) + \delta \|^2}_{\text{decrease } f \circ \phi(q)} + \underbrace{\|q - q_t\|^2}_{\text{small step}} + \underbrace{C^{\mathrm{prior}}(q)}_{\text{motion priors}} \ . \tag{9}$$

The term $\|q - q_t\|^2$ forces small joint steps and thus smooth movements. The term $\varrho \| f \circ \phi(q) - f \circ \phi(q_t) + \delta \|^2$ aims for a decrease in the value function $f \circ \phi(q)$ by a rate δ. The constant ϱ controls the importance of this term with respect to the other terms of C^{MR}. The third term $C^{\mathrm{prior}}(q)$ imposes additional standard costs for joint limits and collisions, i.e. our prior constraints on the motion to be generated.

Iteratively making steps $q_t, q_{t+1}, ...$ with this motion model generates a continuous motion trajectory decreasing the value function $f \circ \phi(q)$. This makes $f(y)$ act like a 1D control variable which should be continuously decreased, resulting in imitation of the desired motions. With our motion model (8) we do not aim to recover the exact sequence of joint states from the demonstrations, but learn to generate motions good with respect to the (latent) value function. Note that the optimization procedure inside the controller does not search for a single joint posture that is a global minimum of the value function in one step, but for a joint sequence that *smoothly decreases* the value function.

The velocity is tuned by the parameters ϱ, δ in Equation (9) which control how fast we decrease the value function. Because of the linearization used in IK, it is reasonable to tune the parameters to make small gradual steps. An alternative motion model can directly make steps proportional to the gradient of C^{MR}. However, our iterative formulation in Equation (8) worked better for us.

The gradient of $f \circ \phi$ with respect to q can be found using the chain rule:

$$\mathcal{J} = \frac{\partial (f \circ \phi)}{\partial q} = \frac{\partial f}{\partial \phi} \frac{\partial \phi}{\partial q} \ . \tag{10}$$

The first term $\dfrac{\partial f}{\partial \phi}$ can be calculated analytically given a parametrization of f. The second term $\dfrac{\partial \phi}{\partial q}$ is the task space Jacobian that can be calculated from our articulated robot simulator for a specific posture q. Because f is a function of task space feature inputs, it can be viewed in itself as a more complex constructed parametrized motion task space.

With this motion model we have the following property, whose proof can be found in the appendix.

Proposition 1 *If $\varrho \to \infty$ then the IK solution q_{t+1} minimizing Equation (9) has the property that the next step $q_{t+1} - q_t$ is approximately proportional to the value function gradient $\mathcal{J}$ in a small region around q_t.*

Note that this proposition holds more strongly the closer we are to q_t and when making small steps; otherwise highly nonlinear value functions can exhibit different behaviour.

3.2 Learning the TRIC Value Function from Motion Data

We defined in the previous section the motion model we use, assuming a value function f existed. We will now explain how we can learn this value function f such that we can generate motion similar to some demonstrated trajectories. We assume that f is a differentiable function parametrized by the vector w – in Section 5 we will specify f concretely.

Suppose the robot observes $D = \{q_t^i, y_t^i, \frac{\partial \phi}{\partial q}(q_t^i)\}_{i=1,t=1}^{N,T}$, a set of N demonstration trajectories of length T. Each trajectory is represented in joint space, projected to feature space $\phi(q_t^i) = y_t^i$, whose Jacobians $\frac{\partial \phi}{\partial q}(q_t^i)$ we also record. [2]

The main assumption we make for learning a value function from data is that the teacher is rational and the demonstrations were optimal with respect to some criteria defined in some motion feature subspace. One way to incorporate this assumption is to design a value function f with the following 3 properties:

(i) The demonstrations move to states of lower value as time progresses: there exists a latent f which acts as a **generative** motion potential.
(ii) Demonstrated teacher motions are discriminated against any other motions: **discrimination**.
(iii) A small set of relevant motion features are selected: **sparsity**.

Property (i) is consistent with our motion model Equation (8), because we want to reproduce the teacher's motion by generating joint states with gradually lower value f. Property (ii) reflects the assumption that the motions of the rational teacher should have lower value function than any random motions, because they are closer to the goal of the demonstration than random motor commands. Property (iii) comes from the assumption that some motion subspace has a good description of the latent goal of the teacher.

To ensure that f has these 3 properties we define the training loss $L(D; w)$ for data D and value function parameters w as

$$L(D; w) = \sum_{i=1}^{N} \sum_{t=1}^{T} (\alpha_n L_n(y_t^i; w) + \alpha_g L_g(q_t^i; w)) + \alpha_w |w|_1 \ . \quad (11)$$

[2] Note that the demonstrations may be from different situations, with objects placed on different locations. Since the features y_i^t may be object relative they cannot be captured from q_t^i alone – which we neglected in our notation. We therefore "record" also y_t^i and the Jacobians $\frac{\partial \phi}{\partial q}(q_t^i)$ for all demonstrations. If it is clear from the context that we are dealing with just one trajectory, we will skip the superscript i and write just q_t instead of q_t^i.

The hyperparameters $\alpha = \{\alpha_n, \alpha_g, \alpha_w\}$ determine the influence of the different loss terms, which we will describe below. Once we have defined the loss $L(D; w)$ we can use gradient-based optimization to optimize it with respect to the parameters w of f. Our training algorithm can handle trajectory data of varying time lengths because the loss function sums over all trajectories and their time steps, but we assume for simplicity that all trajectories are fixed to T time steps.

We will proceed to explain in more detail the loss terms and how they lead to properties (i), (ii) and (iii).

3.2.1 Sparsity via Regularization

Having a motion space with too many dimensions reduces the performance of the algorithm because of the risk of overfitting and poor generalization. TRIC is designed to find a sparse set of task variables which are important for the specific task. The L_1 regularization term $|w|_1$ in Equation (11) forces sparsity in the parameters of the function $f(y)$ and thereby performs feature selection with respect to y.

It is crucial to define a rich set of motion features from which we can extract a compact set of the most important task dimensions using TRIC. The choice of features $\phi(q) = y$ effectively determines what value functions we can learn, and what task spaces we can select as potentially relevant for a given task. It is usually reasonable to take all landmarks and all objects in a scene, and any geometrical relations between them. As long as our feature set y is rich and contains multiple controllable task spaces of the robot body parts relevant for the task at hand, we can expect to successfully imitate with TRIC a large class of motion problems with such motion features. The concrete choice of features will be presented in Section 5.

3.2.2 Discrimination in Feature Space via Loss Term L_n

We want to design the term loss L_n so that properties (i) and (ii) hold. First we make the following observations. (1) The demonstrated trajectories were assumed to be near optimal and should have low values f. (2) All other possible motions should be discriminated from the true motions and have high values f. (3) The trajectory samples ahead in time should have lower f values than those before them.

We use an approximation to model the relation between the teacher demonstration and what we called informally *all other possible motions* or *noise*. For each time t and trajectory i we create a set $\{\tilde{q}_{t,m}^i\}_{m=1}^M$ of

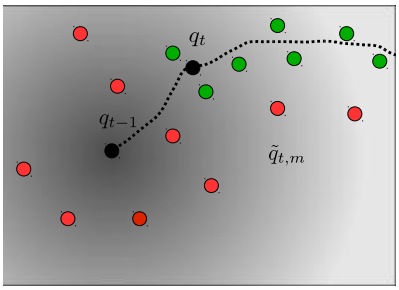


Fig. 2 The noise $\tilde{q}_t$ sampled from a Gaussian centered at q_{t-1}. A few *noisy* samples are shown, colored by their weights $\epsilon_{t,m}$. Low weights are green and high weights are red (for visual clarity the weights are discretized to 2 colors).

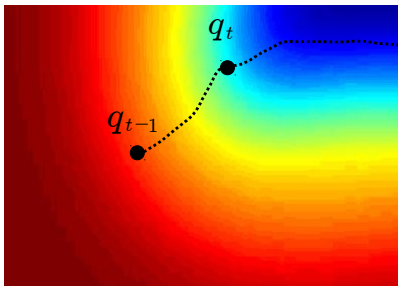


Fig. 3 The loss term L_n pushes down the value function $f \circ \phi(q)$ near the true trajectory samples q_t, and up the cost of the generated noise. Red represents high value.

synthetic *noisy* joint samples which need to be discriminated from the demonstrated trajectories:

$$\tilde{q}^i_{t,m} = q^i_{t-1} + \mathcal{N}(0, \sigma^2) \ , \tag{12}$$

$$\tilde{y}^i_{t,m} = \phi(\tilde{q}_{t,m}) \ . \tag{13}$$

The noisy samples are created by adding Gaussian noise with variance σ^2 to the demonstrated robot joint configuration of the previous time slice. We add noise on q and not on y directly since the feature vectors y lie on a subspace constrained by the robot kinematics and dependencies between the dimensions of y. Thus, we sample in joint space resulting in task space samples lying on a low-dimensional subspace $Y^{\text{valid}} \subset \mathbb{R}^s$ where s is the number of dimensions of y. If we have n joints, this would be a no more than n-dimensional subspace.

We define the distance between a joint state and a joint trajectory interpolated linearly:

$$d_{\text{curve}}(q, \{q_\gamma\}_{\gamma=t}^T) = \min_{\lambda \in [0,1], \gamma \in [t,T]} \|q - q_\gamma - \lambda(q_{\gamma+1} - q_\gamma)\| \ . \tag{14}$$

Further, we specify sample weights for a sample $\tilde{q}^i_{t,m}$ equal to the distance to the true demonstration trajectory *ahead* in time:

$$\epsilon^i_{t,m} = d_{\text{curve}}(\tilde{q}^i_{t,m}, \{q^i_\gamma\}_{\gamma=t}^T) \ . \tag{15}$$

Essentially, this is the shortest distance from $\tilde{q}^i_{t,m}$ to the whole segment of the trajectory from index t until the end T. The time restriction is essential in order to ensure that discrimination will make the correct future states have low f value.

Samples that are near a correct state q_t in the future will have low weights. Samples that are away from the true trajectory in the future will have high weights and their loss contribution will be higher. The way we create these synthetic samples and weight them is illustrated in Figure 2. The term L_n is then defined as:

$$L_n(y^i_t; w) = \sum_{m=1}^{M} \epsilon^i_{t,m} \log(1 + e^{f(y^i_t; w) - f(\tilde{y}^i_{t,m}; w)}) \ . \tag{16}$$

We use the log loss from Equation (7), common in discriminative learning, to penalize noisy samples with low value f. Note that this is in essence a soft version of max margin learning (Tsochantaridis et al. 2005), which is used in other IOC methods.

Minimizing the loss term L_n directly ensures the **discriminative** property (ii). An intuitive interpretation is that we "push down" the value $f(y)$ of the true samples and "push up" the value of the artificial noisy samples, proportional to their weights as in Figure 3, where for simplicity $q = y \in \mathbb{R}^2$.

Minimizing the loss L_n ensures that the **generative** property (i) holds. Minimizing L_n will decrease the value function along the demonstrated trajectory. With enough training data we will be able to generalize the value function to many different situations. Then following the decreasing value function will lead to the generation of the desired motions in novel situations.

3.2.3 Making the Gradient Consistent with the Demonstrations via Loss Term L_g

The last loss term we will discuss is L_g. Its main idea is to force the negative gradient $-\mathcal{J}$ to point in the direction of the difference vector (i.e. velocity) of steps $q_{t+1} - q_t$ of the demonstrated trajectories in joint space. There are several reasons for this design:

- The demonstrations come from a teacher assumed to be rational, so we expect that he moved in the direction of steepest value descent between q_t and q_{t+1}.
- Proposition (1) claims that the optimal next step q_t of the motion rate control model has the property that our controller creates steps in joint space proportional to the negative gradient $-\mathcal{J}$.
- term L_n did not use information from the joint space in its formulation, so L_g will complement it.

The loss term L_g is defined as:

$$L_g(q_t; w) = \frac{\mathcal{J}(q_t)^T}{\|\mathcal{J}(q_t)\|} \frac{(q_{t+1} - q_t)}{\|q_{t+1} - q_t\|} \ . \tag{17}$$

where $\mathcal{J}(q_t)^T$ is the gradient of $f \circ \phi$ evaluated at this particular joint state q_t. Minimizing L_g is equivalent to maximizing the cosine of the angle between $-\mathcal{J}(q_t)$ and $q_{t+1} - q_t$, which is maximal when these are parallel and have angle 0.

The mathematical properties of the gradient (steepest directional derivative) determine the relation of the term L_g to properties (i) and (ii). The value function f decreases along the vector $q_{t+1} - q_t$, thus f decreases in time – the **generative** property (i). Additionally, for a local neighborhood $Q_\gamma := \{q : \|q - q_t\| < \gamma\}$ of q_t for some small enough γ it holds that

$$f \circ \phi(q_t - \gamma \frac{\mathcal{J}(q_t)}{\|\mathcal{J}(q_t)\|}) < f \circ \phi(q) \qquad \forall q \in Q_\gamma$$

This is discrimination between joint states lying in the steepest gradient descent direction around the true trajectory ($q_t - \gamma \frac{\mathcal{J}(q_t)}{\|\mathcal{J}(q_t)\|}$) and any other joint states in the neighborhood Q_γ – the **discriminative** property (ii) for this specific local neighborhood.

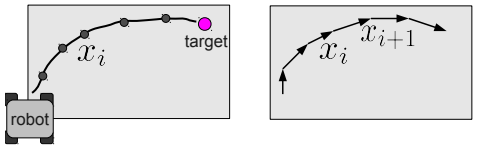
As a technical note, computing $\frac{\partial L_g}{\partial w}$ involves calculating $\frac{\partial \mathcal{J}}{\partial w} = \frac{\partial^2 f}{\partial \phi \partial w} \frac{\partial \phi}{\partial q}$. The first term on the right side can be calculated from f analytically, and the second is the so-called kinematic Jacobian, which we also have in our training data D. This means that minimizing L_g can be done offline, without simulator calls, which simplifies the training architecture and improves performance.

4 Discussion

In this section we will first discuss how TRIC's approach differs from other imitation learning methods. Afterwards we will describe in more detail some properties of TRIC: the loss terms L_n and L_g, and the choice of task space features $\phi(q)$. Finally, we will also discuss some limitations of the current implementation of TRIC, namely that motions with some periodic cyclical structure can not be learned.

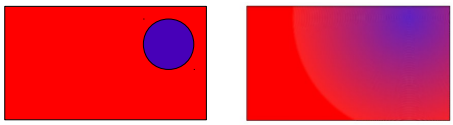
4.1 Goal and Action Imitation

TRIC is capable of imitating behavior in a different manner than DPL and IOC (see Section 2). Figure 4 illustrates these different approaches for imitation in a very simple scenario: a demonstrated trajectory $\{x_i\}$



(a) Demonstrated trajectory $\{x_i\}$.

(b) DPL will learn policy $\pi : \phi(x_i) \mapsto x_{i+1}$.



(c) IRL will learn reward $\phi_{x_i} \mapsto \mathbb{R}$.

(d) TRIC will learn a value function $f : \phi_{x_i} \mapsto \mathbb{R}$.

Fig. 4 A toy example: a demonstrated motion and 3 approaches to imitate it.

reaching a target. There is no information about the reward (negative cost) underlying the motion generation, and no prior knowledge telling the robot that it should always go to the target. There are also some features ϕ that can be used to represent the situation.

We can see DPL, IOC and TRIC as three different conceptual approaches to imitation, using the concepts defined in Call and Carpenter (2002):

- DPL would imitate by memorizing an action to execute in a given state. It would repeat an action without regard of potential outcome – mimicry. Such an approach cannot adapt already seen motions to new unseen situations, so for good performance one would need to provide examples of every possible state and the action for it.
- IOC would learn rewards for each state consistent with a rational teacher. A global planner is then needed to generate a motion that maximizes the accumulated rewards obtained in each state. Because the rewards reflect local properties of states, the planner can generate motion paths that differ from the training sequences, but are optimal with respect to the same rewards – *goal emulation*.
- TRIC is trained so that moving to states of low value imitates the desired behavior. This value function represents aspects of the expected future costs (negative rewards). We can interpret that the value function is a "representation" of the *global* goal (low value), but this value function is also trained to imitate the observations whenever possible. This would be *imitation of both goal and action* in the definitions of Call and Carpenter (2002).

There is a major conceptional difference between the generative models underlying TRIC and IOC. In

IOC, it is assumed that a rational agent maximizes his total accumulated rewards for a whole trajectory. This maximization is achieved by using a planner to find a sequence of motion states with high accumulated rewards. TRIC, on the other side, assumes that a rational agent needs to move in the direction of decreasing value function at each step, and thus a local reactive controller can work well with this assumption.

4.2 Discussion of the Value Function and Loss Terms

In Section 3.2.2 and 3.2.3 we defined the terms of the loss used for training TRIC, and motivated why they lead to desired motions with our motion model. Here we will discuss some further aspects of these terms.

4.2.1 Discussion of the Loss Term L_n

The L_n Equation (16) has similarities with IOC and Equation (4):

- the expected state features $\mu(\pi)$ correspond to our motion features y
- the comparison of the true policy π^* to the other policies π is similar to comparing the true trajectory sample y_t to the noisy data points $\tilde{y}_{t,m}$: both lead to terms in the training loss that are large when the true trajectory is not preferred to any other motion
- the scalable margin $\mathcal{L}(\pi^*, \pi)$ corresponds to the weights $\epsilon_{t,m}$: both quantify the penalization in the training loss terms. It is larger when a preferred false trajectory is away from the true trajectory

Another comparison we can make is with behavioral cloning and the method of Ratliff et al. (2009). Minimizing loss L_n maximizes the likelihood of the training data (trajectories q_t) and lowers the likelihood of the noise around them. A difference between TRIC and behavioral cloning is that TRIC does not have any explicit action space and uses instead the value function f itself to generate motions in the direction of lower f.

A comparison can also be made between discriminative learning in ML and the term L_n. The usual approach to discriminative learning as in Equation (6) would require finding the most offending false answer and discriminating it from the true answer. We do a similar discrimination (where value function corresponds to energy) between the true sample y_t and the noisy data points $\tilde{y}_t$. However, we use a discrete set of such samples to approximate the space of these samples in a small neighborhood of the last joint state q_{t-1}. This is faster than looking for the most offending false answer, but potentially inaccurate if there are not enough samples.

4.2.2 Discussion of the Loss Term L_g

It can be shown that L_g is related to Howard et al. (2009) and DPL as formulated in Equation (3). Consider the normalized gradient $\frac{\mathcal{J}(q_t)^T}{|\,\mathcal{J}(q_t^i)\,|}$ as control policy $\pi(q_t)$ and the control signals $u_t = \frac{q_{t+1}-q_t}{|\,q_{t+1}-q_t\,|}$, normalized to constant length. Then both the DPL loss in Equation (3) and L_g loss in Equation (17) will be minimized when $\pi(q_t)^T u_t = 1$ for all joint states q_t. When policies and motor commands are normalized, minimizing projection discrepancy becomes equivalent to minimizing the angle between motor command and policy prediction. The relation of DPL and L_g is also a strong indication that the term L_g leads to a value function robust to unseen constraints (see the experiments in Section 5.3.4), and this robustness reminds of the method of Howard et al. (2009).

Khansari-Zadeh and Billard (2010) present an error function for imitation learning with two terms: the difference between magnitudes of velocity vectors of the demonstration data and controller prediction, and the difference between the directions of these velocity vectors. Our loss term L_g is maximizing the cosine of the angle between the negative gradient $-\mathcal{J}$ and the velocity vector $q_t - q_{t-1}$. In Proposition (1) we showed how $\mathcal{J}$ is influencing the direction of the joint steps created by the motion rate controller, and thus the direction of the observed velocity vector influences the motion created by our model. However, the velocity magnitude information is lost by TRIC because of an additional scaling factor in our motion controller, see Proposition (1). If the velocity information was crucial for the task at hand, the required additional tuning (see Section 5.2.3) can be a drawback for TRIC.

4.2.3 Properties of the Attractors of the Value Function

A minimum of f acts as an attractor to the motion generation system defined in Equation 8. We can analyze its stability properties using notions from the classical Lyapunov Stability theory (Slotine and Li 1991), applied to motion rate control. If a motion system converges to some equilibrium point regardless of the starting point it will be called asymptotically stable.

Definition 1 The motion system $x_{t+1} = x_t + g(x_t)$ is asymptotically stable at the point x^* if starting from any x_0 it converges asymptotically to x^*, i.e.

$$\lim_{t\to\infty} x_t = x^* \ .$$

Proving such a property for a motion system is useful, because it signifies that the motions can reach their

targets robustly with respect to any perturbations and random starting conditions (Perkins and Barto 2002; Khansari-Zadeh and Billard 2010). There is the standard Lyapunov Stability theorem that states a condition when the asymptotic stability property holds:

Theorem 1 *The motion system $x_{t+1} = x_t + g(x_t)$ is asymptotically stable at the point x^* if a continuous function $V(x)$ – continuously differentiable and changing over time as $\triangle V(x)$ – can be found such that:*

$$\begin{cases} (a) & V(x) > 0 & \forall x \neq x^* \\ (b) & V(x^*) = 0 & \\ (c) & \triangle V(x) < 0 & \forall x \neq x^* \\ (d) & \triangle V(x^*) = 0 \ . & \end{cases}$$

We now proceed to claim that TRIC is asymptotically stable under some assumptions, and the proof is in the appendix.

Proposition 2 *Suppose we have trained TRIC on a single trajectory $\{q_t, y_t\}_{t=1}^T$ and that $f \circ \phi(q_T)$ is a minimum of the value function. Additionally, we generate motion with $\varrho \to \infty$, i.e. very high weighting of the value function. Then the motion generated by the model in Equation (8) fulfills the conditions of Theorem 1 and is thus asymptotically stable at the attractor subspace $Q' = \{q' : \phi(q') = \phi(q_T) = y_T\}$.*

This proposition tells us that we have as an attractor a whole joint subspace Q' to which our motion generation method will converge given enough time. The attractor is not a single state but a subspace, because redundancy of the joint space allows for multiple different robot joint configurations q' to have the same task features y_t.

The assumption that $f \circ \phi(q_T)$ is a minimum is reasonable because of the construction of the loss term L_n to discriminate the future states of the trajectory and make them have a lower value. It was also verified empirically to hold in our data, see Section 5.4.

We can not prove convergence properties of TRIC in a realistic scenario with multiple trajectories and ϱ not going to infinity. If we have extreme cases, e.g. an obstacle blocking all paths to the target and pushing the robot away, no performance of the system can be guaranteed because obstacle avoidance will influence the behavior more than the task-related value function. However, our experiments in Section 5 show empirically that motion generation using the learned value function is robust and stable.

4.3 Limitations of the TRIC Model: Periodic Motion

The value function f acts as a potential over states $\phi(q_t)$ and TRIC is designed with the premises that (1) f decreases monotonically across the trajectory as it goes ahead in time and that (2) f has some minima to which the system goes and stops. However, this is not the case in tasks that have a looping periodic aspect: such motions cannot be generated by TRIC. An example of this issue can be the task of following a demonstrated circular trajectory of the robot hand around a central point, and looping continuously the same motion.

Another approach to modeling the value function would be to add to the features for time t information from the previous state of the robot q_{t-1} as information and learn a value function over data $f \circ \phi(q_{t-1}, q_t)$. This would allow to imitate the observed trajectories in a more sophisticated way and thus solve the circular movement problem indicated above by having an attractor in a space where q_t is curved from q_{t-1} in an appropriate way, but testing this is left to future work.

For comparison, some imitation learning methods can handle periodic motions quite well. For example, the DMP research has multiple examples of good modelling of periodic motions, see e.g. Schaal et al. (2003); Ude et al. (2010). DPL methods that predict the next action can also learn periodic motions by simply predicting appropriate consecutive motion states by regression. Potential field methods will need special adaptations to handle periodic motions, similar to TRIC. And methods like Ratliff et al. (2006) that use an A* planner to find an optimal path to a target will not be able to handle periodic motions.

5 Experiments

We examined several robot motion tasks in simulation to see the performance of TRIC. The scenario we examined in most detail is grasping of spherical objects. We also investigated grasping of cylinders in the presence of other obstacles, and a scenario where the robot has to open a box. This section is organized in the following way. In Section 5.1 we describe our robot experimental setup and why the grasping task is interesting for imitation learning. Afterwards in Section 5.2 we define the parametric model for the value function f. We also define the motion features y and the settings of the motion rate controller. In Section 5.3 we present numerous experiments and results for grasping and box opening with TRIC. In Section 5.4 we analyze the **discriminative** and **generative** aspects of TRIC – properties (i) and (ii) from Section 3.2. Finally, in Section 5.5 we an-

alyze the extracted **sparse** motion features, property (iii).

For all results, we used our simulated robot setup of the Schunk LWA 3 arm with 7 Degrees of Freedom (DoF) and Schunk SDH hand with 7 DoF, i.e. a joint configuration space $q \in \mathbb{R}^{14}$.

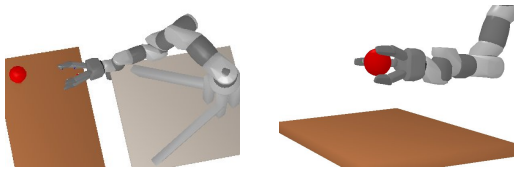
5.1 The Grasping Task

Grasping is essential for robot manipulation of everyday objects. It is a complex manipulation task that requires control of multiple robot body parts with respect to an object. This means that there are many possible task spaces relevant for control. Grasping has received attention in the context of learning by demonstration and robotics and is a good example of how many different motion representations have been designed by experts for grasping. Tegin et al. (2009) acquire grasps from human movements and repeats them in joint space. Kroemer et al. (2009) explore grasps in a parametrized 6D space of position and orientation of the gripper, and learns a value function with regression. Kroemer et al. (2010) use the hand fingertips for grasping within the DMP framework. Gienger et al. (2008) create a task manifold of presampled grasp positions and orientations and a motion potential towards them. It is an interesting challenge to extract the important task spaces from data and learn good controllers with as few prior assumptions as possible for good grasping, and this is what we will demonstrate with TRIC.

The basic scenario setup for grasping consists of the robot and a target object. We restrict the grasp target to be a sphere or cylinder to simplify the setup, but even with such simple geometry the grasp task requires complex robot motion. For grasping demonstration we use the method of Dragiev et al. (2011), see Figure 5. The controller of Dragiev et al. (2011) is a fully autonomous grasping controller, but it has been designed with expert knowledge of what is a good grasp. It has several hard-coded phases, each of which controls the motion different robot body parts and gives them different relative importance. All this information is hidden from TRIC, which has to recover the behavior just from observation. We visualize the joint angles for one sphere grasping trajectory in Figure 6.

5.2 TRIC Setup: Motion Representation and Value Function Model

The description of the TRIC method in Section 3 left the choice of task space ϕ and value function model f to be specified concretely depending on the task at hand.



(a) First approach the red target object and open the fingers

(b) Finally align the fingers close to the object's surface

Fig. 5 Illustration of a grasping motion that the robot has to learn.

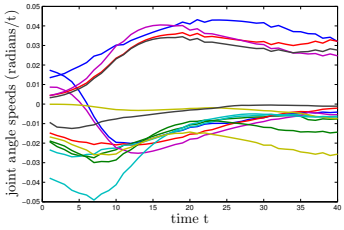


Fig. 6 A sphere grasping trajectory in joint space for $T = 40$ time steps. Each of the 14 dimensions of the joint velocities $q_{t+1} - q_t$ is plotted as a line.

We will describe how this was chosen concretely for our grasping task.

5.2.1 Selecting a Rich Set of Task Spaces

The effectiveness of TRIC depends on selecting a rich set of task space motion features that seem reasonable for a certain task. For our experiments we decided to use a rich geometrical representation, namely the pairwise distances between a set of important landmarks, defined on the robot body and external objects in the environment. The feature space $y = \phi(q)$ consists of $\hat{b}$ pairwise landmark relative positions p_i and their norms $d_i = \|p_i\|$ for each landmark pair $i = (j_1, j_2)$:

$$y = (p_1, d_1, ..., p_{\hat{b}}, d_{\hat{b}}) \in \mathbb{R}^{4\hat{b}} \tag{18}$$

Our approach to task space selection needs to define the set of landmarks and the geometric motion features, representing a prior on motion representations which are potentially suitable for the task at hand.

For the grasping experiments we defined a set of 8 landmarks on the robot and the grasp target object $A = \{a_i\}_{i=1}^{8}$, shown in Figure 7. This includes the center of the target object a_1, the three fingertips a_2, a_3, a_4, the three lower parts of the fingers a_5, a_6, a_7, and the palm center a_8. The feature space consists of pairwise landmark positions and their norms: $y = \phi(q) \in \mathbb{R}^{224}$. We preprocess each dimension of y by rescaling it in $[0, 1]$. Some features are redundant due to the specific

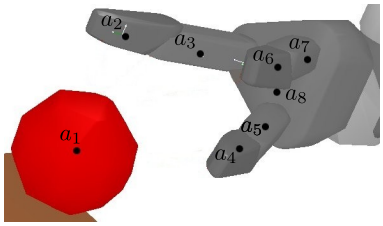


Fig. 7 The features y are defined using the landmark set $A = \{a_i\}_{i=1}^{8}$, indicated by black dots.

geometry of the landmarks. E.g., $d_{i,j} = d_{j,i}$ because the norm of a vector does not change as it is rotated in some frame. We remove the redundant features using correlation as a measure (Haindl et al. 2006), resulting in final motion features $y \in \mathbb{R}^{150}$.

This approach to modeling task spaces with information between geometric bodies in a scene is related to the interaction mesh approach in animation, see Ho et al. (2010).

5.2.2 Parametrization of the Value Function

The description of the TRIC algorithm so far assumed that the value function f is a differentiable function parametrized by a vector w. The definition of loss term L_g makes it also necessarily to condition the gradient of the value function, so this makes many standard regression methods (e.g. SVM) unsuitable for TRIC. Concretely for our experiments we chose a neural network with K sigmoid units *plus* a linear term:

$$f(y; w) = z^T W_2 + y^T W_3 \ , \tag{19}$$

$$z = \frac{1}{1 + e^{W_1 y}} \ . \tag{20}$$

The parameters w consist of $W_1 \in \mathbb{R}^{K \times s}$, $W_2 \in \mathbb{R}^K$, $W_3 \in \mathbb{R}^K$ and $z \in \mathbb{R}^K$ is the hidden layer; $s = 150$ is the dimensionality of y. This neural network is fully differentiable and fits well to our requirements.

The complexity of this neural network model is $O(sK)$ for evaluation of $f(y)$, $\frac{\partial f}{\partial w}$ and $\frac{\partial f}{\partial y}$. The second partial derivative $\frac{\partial^2 f}{\partial y \partial w}$ has complexity $O(s^2 K)$, which depends quadratically on s.[3] We trained our models for 100 iterations with the BFGS method from the MATLAB Optimization Toolbox. The training time scales with K: for a linear model $K = 0$ it takes less than a minute, and 30 minutes for $K = 30$.

[3] Here we write $\frac{\partial f}{\partial y}$ instead of $\frac{\partial f}{\partial \phi}$, because $y = \phi(q)$.

5.2.3 Parameter Settings of the Motion Rate Controller

We use Equation (8) for motion rate control using the learned costs $f \circ \phi(q)$. The magnitudes of the value function f can be quite different, and this directly influences the number of steps required to reach the value function minima. The parameter δ influences how many steps the controller needs to generate a motion until the final steps of the demonstrations. As a practical way to set it up, we try to tune it standardly so that around 500 steps are necessary to go from the start to the end of the demonstrations

$$\delta = \frac{500}{N} \sum_{i=1}^{N} f(q_t^i) - f(q_T^i) \ .$$

We use also standard additional constraints on collision avoidance and joint limit avoidance coded in the term C^{prior}, with weights high enough to ensure that the IK controller stays collision free and within joint limits. The values of the parameter ϱ should be set to ensure the proper balance for the task at hand between the prior constraints and the task specific constraints.

5.3 Experimental Results: Performance of the Learned Controller

For quantitative comparison we inspected grasp costs of TRIC with different settings. We use the errors of the task variables defined in Dragiev et al. (2011) as a cost metric to validate how good the grasps of the TRIC controller are. The teacher demonstrations were good with respect to this cost, but it was hidden from TRIC during the training. The cost metric evaluates the final grasping posture of motion with a number between 0 and 1, where values below 0.25 are good grasps.

The default values of the loss parameters from Equation (11) are $\alpha_n = 1, \alpha_g = 1, \alpha_w = 0.001$, and the other parameters are $M = 60$ samples, $N = 27$ trajectories, $K = 30$ sigmoids for the model of f. We generate the M random samples by randomly sampling collision-free joint configurations with a normal distribution with $\sigma = 0.05$, corresponding to 0.05 radians joint angle standard deviation.

For training we generate a dataset by translating the position of the target object a_1 in a regular grid of $50 \times 40 \times 40$cm in front of the robot, taking 3 positions in each grid dimension, leading to $N = 27$ different settings and training trajectories. We would call each of these different settings "situations", since each a different initial setup requires different motion for a good grasp. Each grasp movement is generated for 5 seconds,

and we sampled $T = 40$ time steps from it, once every 125ms. The target is grasped faster when the object is closer to the robot, so some of these motion trajectories grasped the object sooner or later than others.

5.3.1 Experiment 1 - Influence of the Loss Parameters on Sphere Grasping

For testing the grasp costs of the controller learned with TRIC we created a new set of 15 trajectories on random positions within the training grid and evaluated the grasping costs of different TRIC models with 500 steps taking 5 seconds for execution, one step every 10ms. This is a faster control rate than the sampling rate of the training set. We use the relaxation of not following the exact velocities of the teacher, as discussed in Section 4.2.2.

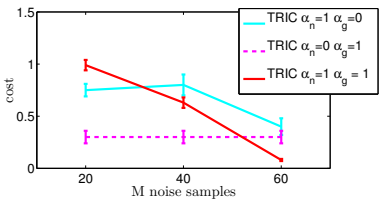
We tested 3 variations of the TRIC model by changing the parameters α_g and α_n controlling the weights of loss terms L_g and L_n. Figure 8(a) shows how the performance of TRIC changes with respect to the number of noisy samples M. Figure 8(b) shows how the number of test trajectories N influence the performance. We make the following observations:

- With only 3 trajectories TRIC could learn grasping well - efficiency in terms of N.
- The parameter M seems more critical than N. Since the minimization of the TRIC loss is linear in terms of M and N, this shows that to speed-up training it is better to have a few trajectories N but create many noisy samples M.
- $M = 60$ samples were enough to approximate the local neighborhoods in the subspace of valid robot configurations (see Section 3.2.2), and we expect no further gains by increasing M to larger values.
- The best parameters of TRIC are $\alpha_g = 1$ and $\alpha_n = 1$, indicating that both loss terms L_g and L_n are important to learn the value function.

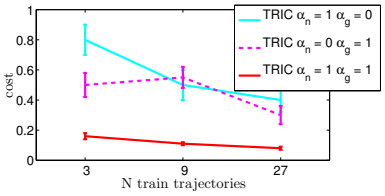
A possible explanation for the benefit from including both terms L_n and L_g can be the different information they represent from the raw demonstration data. In the case of L_n this information consists of the true and noisy generated samples in the high dimensional task space y, and learning with this information forces the value function f to take appropriate values at these samples. For L_g the information is the direction of the true trajectory steps in configuration space q, and learning with this information conditions the gradient of f to have this direction. The experimental results indicated that is is indeed beneficial to use both types of information. The two spaces, y and q, are connected by the nonlinear kinematic map $y = \phi(q)$, so it seems reasonable that both spaces should play a role for motion imitation.

method	grasping cost
Teacher	0.06 ± 0.01
TRIC	0.08 ± 0.01

Table 1 Sphere grasping results: TRIC compared to the teacher on 15 test situations.



(a) The effect of changing the number of noisy samples M on TRIC.



(b) The effect of changing the number of training motions N on TRIC.

Fig. 8 The effects of changing training setup parameters on TRIC: noisy samples M and training trajectories N are varied.

joint speeds $\lvert u_{t[k]} \rvert$	0.009 ± 0.001
training set errors $\lvert u_{t[k]} - \pi(q_{t[k]}) \rvert$	0.002 ± 0.001
test set errors $\lvert u_{t[k]} - \pi(q_{t[k]}) \rvert$	0.006 ± 0.001

Table 2 Absolute values in radians of the joint speeds $\lvert u_{t[k]} \rvert$, and the training and test set errors $\lvert u_{t[k]} - \pi(q_{t[k]}) \rvert$ of DPL, averaged over the 14 joint dimensions $k \in \{1, 14\}$ and over all situations.

Overall, TRIC performed close to the teacher heuristic from Dragiev et al. (2011), as we summarize in Table 1, showing the mean and standard deviation of the costs.

5.3.2 Experiment 2 - Comparison of TRIC with DPL in Sphere Grasping

In a second experiment we implemented as baseline a simple DPL method for comparison with TRIC, using the same training set as in the previous experiment. We used the pairwise geometrical information feature y as the state space x in the DPL notation, $u_t = q_{t+1} - q_t$ for control u_t, and a neural network with 60 sigmoids to learn π, a regression problem with least-squares loss as in Equation (2).

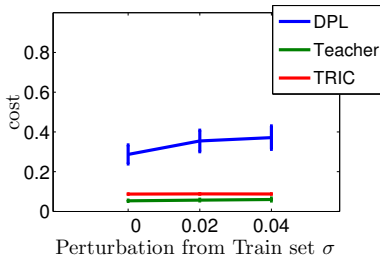


Fig. 9 Comparison of TRIC to DPL on test sets with increasing "distance" between training and test data of $\{0, 0.02, 0.04\}$ meters.

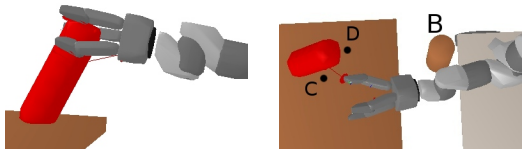
Table 2 shows the errors of the DPL policy mapping $\pi : q_t \mapsto u_t$ on the training and test sets. The much larger test set error indicates that more data is necessary to learn a well generalizing mapping from world state x to motion command u_t.

After we trained the policy π, we implemented an IK controller for DPL imitation using the following IK motion generation scheme:

$$C^{\text{DPL}}(q, q_t) = \|q - q_t\|^2 + 10^3 \|q - q_t - \pi(q_t)\|^2 + C^{\text{prior}}(q) \ ,$$
$$q_{t+1} = \underset{q}{\text{argmin}}\ C^{\text{DPL}}(q, q_t) \ .$$

To test DPL performance more quantitatively we defined a new validation set, consisting of (a) 30 scenarios from the original training set, (b) 30 scenarios modified from the training set with added random translation to the target position sampled from a Gaussian with 2cm standard deviation, and (c) 30 scenarios created analogously with 4cm standard deviation. Figure 9 shows how the results degrade for DPL as the validation scenarios become remote from the training samples, whereas TRIC remains robust.

The relatively poor performance of DPL with joint space Q for the motion commands is to be expected, because commands in joint space have difficulties in generalizing to translations of the objects in the workspace: a linear translation of the target results in complex nonlinear change of the joint space trajectory. If the robot task was just point reaching, DPL could be improved by selecting a task space like $Y_{\text{target}} \in \mathbb{R}^3$, endeffector coordinates, which can easily generalize to translations of targets. For grasping there are no such obvious solutions. Even if we take $\phi(q_{t+1}) - \phi(q_t)$ as motion command in the high dimensional space of $y \in \mathbb{R}^{150}$, as defined in Section 5.2.1, we could not get reasonable performance. The problem was that the machine learning prediction task becomes too complex in this case - a mapping of dimensionality $\mathbb{R}^{150} \mapsto \mathbb{R}^{150}$.



(a) A robot grasping pose for a cylinder.

(b) Grasping the target while avoiding obstacle.

Fig. 10 Implicit obstacle avoidance and cylinder grasping with TRIC.

method	grasp costs
Teacher	0.07 ± 0.01
TRIC $K = 0$	0.19 ± 0.02
TRIC $K = 30$	0.09 ± 0.02

Table 3 Result for cylinder grasping on a set with $N = 10$ motions and default hyperparameters, TRIC with linear and nonlinear models compared with the teacher performance.

5.3.3 Experiment 3 - Cylindrical Objects

In a third experiment we tested grasping of a more complex object – an elongated cylinder with 30cm height and 5cm radius. We created a new training set of $N = 10$ demonstrations, by rotating the cylinder on its Y-axis randomly by an angle in $[0, \pi/2]$. We then trained TRIC with the default parameters and tested the learned f on grasping of cylinders rotated differently than in the training set, see Figure 10(a). As can be seen in Table 3 the performance of TRIC with nonlinear model $K = 30$ is similar to the Teacher. Linear TRIC ($K = 0$) was worse than the nonlinear one, thus, more complex motion policies require non-linear models for f. It is also worth noting that as few as $N = 10$ trajectories of length $T = 40$ were enough to train TRIC – efficiency in terms of demonstrations required.

5.3.4 Experiment 4 - Obstacle Avoidance as Motion Constraint

In a fourth experiment we examined how TRIC reacts to an obstacle B in its path to the target. Because our motion model (Eq. 8) includes a prior cost term $C^{\text{prior}}(q)$ for avoiding obstacles, the motion controller can handle such cases robustly. In our evaluation the obstacle B appears *only* in the validation set, not in the training set. TRIC can generalize to these previously unseen obstacles: the motion rate controller tries to stay out of collisions and will adapt to a grasping pose that is compatible with this constraint, see Figure 10(b). The straight approach to grasp pose D is no longer feasible, because it will collide with B. Another grasping pose C can be found, one of the many grasp poses with low value f in the subspace of good grasps,

see also Figure 15 for additional analysis of this space. Because it is also reachable without collisions, C was preferred to D. This is an illustration of the generalization ability of TRIC. DMP methods also have some ability to adapt to unexpected perturbations. DPL is less robust: its learned behaviour is to repeat the seen trajectories and it can not adapt them to never before seen obstacles if they require deviation from the usual motions.

Clearly the local collision avoidance behavior of our controller has limitations – in a more complex world cluttered with obstacles reactive avoidance can get stuck in local minima, a drawback common to all local reactive controllers (see Siciliano and Khatib 2008).

5.3.5 Experiment 5: Box Opening

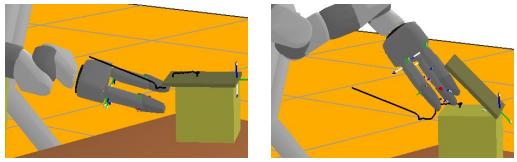
Besides the various grasping tasks we also investigated a completely different task: manipulation of an articulated object, a box with a movable cover. The task is considered achieved if the cover moved up by more than 0.3 radians. To create demonstrations we used the motion planner AICO (Toussaint 2009) as a teacher. The planner calculates a trajectory with $T = 80$ time steps that minimizes a trajectory cost function. We handcrafted a box opening cost function that leads to motion with two distinct phases: reaching the cover below its edge, and then moving the hand up and toward the box center in an arc to gradually open the box cover, see Figure 11. It is an additional complication that the robot should avoid collisions with the box but can touch the edge of the cover with its fingers to move it.

We created random situations in this scenario by translating the box, rotating it, and also changing the size of the box (a cube with side between 10 and 25cm). The training set consisted of $N = 10$ recorded opening motions, and testing was done on other 25 random situations. Note that the random scenario generation method we use creates quite different situations, so with only 10 training situations the controller has to learn to generalize in order to have a chance to succeed on the test set. We used the same model for TRIC value function f as in Section 5.2.2. The features we used were based on a set of landmarks (see Section 5.2.1). We used a set of 5 landmarks $A = \{a_i\}_{i=1}^{5}$, with landmarks on the box, on the cover, on 2 of the robot fingers and on the immobile robot base. This made for task space features $y = \phi(q) \in \mathbb{R}^{80}$.

Table 4 shows the performance of TRIC and two variations of DPL, predicting controls in the joint space Q and the finger in box frame 3D space, see Figure 12. TRIC was very good on the test set with 24 successes out of 25 tries, which shows how robustly it can deal with the test set situations. DPL failed to imitate box opening in the joint space, and was slightly better but still poor in the specially selected task space of the finger coordinates in the box frame.

method	success
TRIC	24
DPL joint space	2
DPL finger space	9

Table 4 Successful box openings out of 25 test trials: TRIC and DPL in two different spaces. The teacher achieved success in all 25 trials.



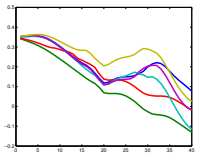
(a) First, move hand toward box edge.

(b) Second, raise hand to open the cover.

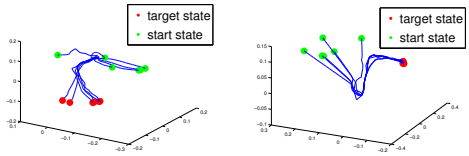
Fig. 11 The demonstrated box opening task and its 2 phases.

Figure 12 shows 5 example motions (opening different boxes) projected to 3 different task spaces, and illustrates why discovering task spaces is useful for the box opening task. The trajectories look quite different in joint space (Figure 12(a)), which is a typical problem for configuration space imitation. In the space of the box in the 3D frame of the fingers (Figure 12(b)) there is some structure, but the variation in the set of final positions (red dots) is still too large to allow effective control. If we take, however, the space of the finger coordinates in the box frame (Figure 12(c)), we see a clear structure of the fingers going to an attractor trajectory towards the compact set of final states (red dots), regardless of the different start states (green dots). TRIC discovered the useful task spaces, which explains the good results in Table 4.

This experiment shows clearly the qualities of imitation with TRIC. Creating a good box opening trajectory with the planner required 3 seconds just for planning before taking any action. We managed to train TRIC with a success rate close to the teacher's, and our controller is fast and reactive. Thus, we not only learned robustly the task without prior knowledge, but also gained the ability to execute it faster than the teacher method. The learned value function encodes the behaviour for which originally a much more complex and slow motion generation system (with a planner and cost function) was required. Learning from demonstrations selected the most representative geometric characteristics of the box opening task, and these essential features allow the reactive greedy controller (simpler than a planner) to perform complex motions.



(a) First joint angle



(b) Box in finger frame (c) Finger in box frame

Fig. 12 Plots of 5 robot trajectories in 3 different task spaces, illustrating the importance of correct task space selection.

5.4 Analysis of the Learned Latent Value Function

So far we focused on the performance of the TRIC method for motion generation. Another way to examine TRIC is to look at the structure of the grasping data uncovered by our method: the properties of the value function after training and the representation of the desired grasping poses. The results in this section come from analysis of the first grasping task defined in Section 5.3, and use for data the 27 training situations and 6 of the random test situations defined there, and their respective motion trajectories.

5.4.1 Examining the Value Function f and its Gradient after Training

The TRIC models were trained robustly with respect to the different parameters. Figure 13 shows that we successfully learned a value function f always decreasing in time, an invariant property of the demonstrated sphere grasping motions. We can also notice in this figure that we have identical behavior of this value function on the 27 training situations and the 6 test situations (indexes 28 to 33 in the figure). This is an indication on how our model generalizes from the training data to unseen novel situations in the same domain.

In Figure 14(a) we illustrate how TRIC could align the gradient $\mathcal{J}$ to the trajectory relative steps $q_{t+1} - q_t$. Most of the data points had a value of $L_g(q_t^i)$ near -1, indicating that TRIC successfully learned a value function with this property. However, near the final time steps $t > 30$ this value worsens a bit. We have a possible explanation for this effect. Near the final phases of the motion the collision cost $g_{\text{collision}}$ (a term of C^{prior}) increases, as shown in Figure 14(b). This increase is due to the fact that the robot fingers close on the object surface near the end of a good grasp. The collision costs increase together with the magnitude of the collision gradients. As a result, the value function is not directly decreased in the steepest descent manner, but in a way combined with collision avoidance, resulting in a less-steep value function decrease.

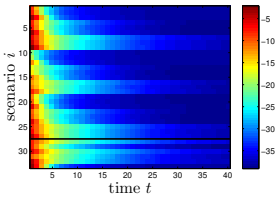
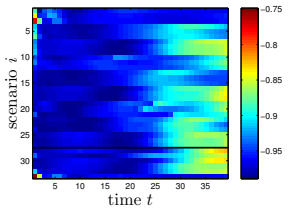
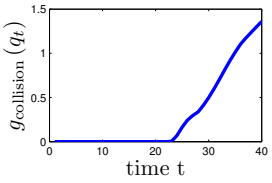


Fig. 13 TRIC learns a value function $f(y)$ decreasing for all observed trajectories y_t^i for time steps $t \in \{1, ..., 40\}$ and situations $i \in \{1, ..., 33\}$. Blue indicates low value.



(a) Cosines of the gradient angles for all situations and trajectory steps. Blue values are desired in the loss criteria.



(b) A plot of the collision costs of a grasping motion: near the end the collisions rise.

Fig. 14 The value $L_g(q_t^i)$ reflecting cosine of angle between $\mathcal{J}$ and $q_{t+1} - q_t$ minimized by loss term L_g for all observed trajectories y_t^i for time steps $t \in \{1, ..., 40\}$ and situations $i \in \{1, ..., 33\}$.

5.4.2 The Subspace of Good Grasps

Another way to analyze the effect of training TRIC is to look at the hidden layer $z = \frac{1}{1+e^{W_1 y}}$ of the neural network model for value f as defined in Equation (19), see Montavon et al. (2011). Figure 15 shows a low dimensional kernel PCA visualization, using a Gaussian kernel on the raw features y and the learned hidden (latent) representations z of the 27 training trajectories. As we showed in Figure 13, the trained value function decreases monotonically in time, so the color is also an indicator of the time progress of the sample trajectories,

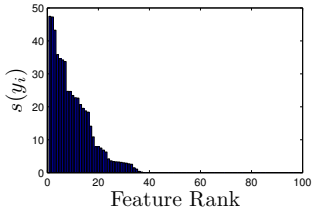


Fig. 16 Effect of L_1-regularization: the motion features sorted by score $s(y_i)$ from the coefficients of the TRIC model. In our notation $s(y_i)$ is the score of feature i.

$t = 1$ is red and $t = 40$ is dark blue. In Figure 15(a) the set of goal states (colored blue) is scattered in a region and seem different depending on the different grasp posture. With such representation it is not easy to define what is a good grasp. Figure 15(b) shows that the training of TRIC produced latent z values such that all goal states are close to one another, much closer than in the first plot. Thus the trained neural network value function has found a compact representation of the subspace of good grasps. This representation allows TRIC to switch between different grasps when required, as in the experiment in Section 5.3.4.

5.5 Analysis of the Motion Features: Retrieving Relevant Task Spaces

We just showed the quality of the controller and value function learned by TRIC by examining the way it grasps. Another aspect of TRIC is the contribution of different features to the learned value function, which can lead to a better explanation of what exactly the robot learned from the teacher. This section also uses for all analysis 33 training and test motion trajectories, as in the previous Section 5.4.

5.5.1 Sparsity of Features Contributing to the Value Function

To get an insight into the feature selection done implicitly by the loss minimization of Equation (11), we defined a score for the contribution of each feature to the value function f, as defined in Equation (19). It is defined as a weighted sum of the absolute values of coefficients of the model of f :

$$s(y) = |W_1^T||W_2| + |W_3| \ .$$

The notation $|.|$ stands for absolute value of the matrix elementwise. This is a heuristic to see how much each feature contributes to the value function f. In Figure 16 we display these scores, using one of the trained models for the sphere grasping task. It can be seen that the L_1-regularization selected about 38 of these features for the cost f. We inspected these top rated features, and found out that the ones with highest scores were these coding distances from the fingers to the target object.

5.5.2 Analysis of the Gradients of the Value Function

Another interesting way to extract structure using TRIC is to look at the gradients $\frac{\partial f}{\partial y} \in \mathbb{R}^s$ evaluated at each data point y_t^i. This allows to analyze the time progress of the feature contribution to the activation of f. We define the average gradient $\bigtriangledown f_t \in \mathbb{R}^s$ for a time step t:

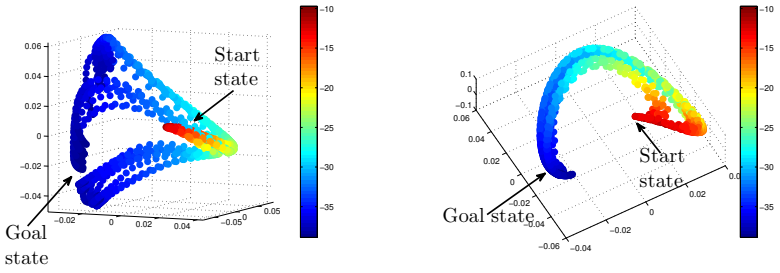
$$\bigtriangledown f_t = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial f}{\partial y}(y_t^i) \ .$$

We are averaging over $N = 33$ situations the values of all evaluated gradients for a fixed time step. If we look at the individual dimensions of the vector $\bigtriangledown f_t$ this gives us the average contribution for time t of feature j to the value function. Note that the motions are not aligned perfectly in time, so this adds some distortion since $\bigtriangledown f_t$ averages for fixed time steps. A better analysis could be achieved by applying some form of Dynamic Time Warping for alignment in time (Myers and Rabiner 1981).

Figure 17 displays the values of $\bigtriangledown f_t$ for the most representative 20 dimensions by value of $s(y)$. We see how the contribution of each feature changes over time. We see that in different time steps different features have bigger contribution to f. This is due to the nonlinear model we used for f with $K = 30$ sigmoid units. A linear model (with $K = 0$ nonlinear units) for f in Equation (19) would have constant gradients for every time step, equal exactly to W_3.

The detailed view in Figures 17(b) and 17(c) shows the raw value and $\bigtriangledown f_t$ just for the 8th best feature, which has the geometric information of $p_{3,1}^y$, the y distance between the finger a_3 and the target a_1. We see that the gradient starts positive in the initial steps, so the TRIC controller will decrease the value of $p_{3,1}^y$ in order to decrease $f(y)$. As time progresses after $t = 10$ the sign of $\bigtriangledown f_t$ changes and now $p_{3,1}^y$ will be decreasing much more slowly.[4] Finally after time $t = 25$ the distance stabilizes at 0.08 meters. In Figure 14(b) we showed how the collision gradients become active only after $t = 25$, so the effects we observe after $t = 10$ are because of the value function properties.

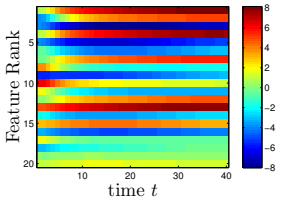
[4] It is still decreasing despite the gradient sign change because of other features coupled geometrically to $p_{3,1}^y$.



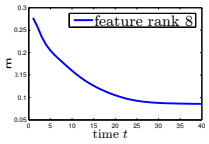
(a) Kernel PCA using Gaussian kernel on raw features y. Large goal state region.

(b) Kernel PCA using Gaussian kernel on hidden neural network layer z. Compact goal state representation.

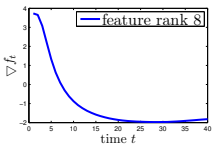
Fig. 15 Kernel PCA visualization of the trajectory data. The color indicates the value function f, which is inversely correlated with the time index t after proper training.



(a) Average gradients for the top 20 ranked features.



(b) Plot of the 8th ranked feature, distance in meters.



(c) Plot of the 8th ranked feature gradient.

Fig. 17 Plots for the grasping data of average gradient $\bigtriangledown f_t$ vs time t, and a detailed view of the 8th ranked feature. All values averaged over 33 different trajectories.

6 Conclusions

In this article we presented a novel method for learning a sparse discriminative value function from demonstrations, finding relevant task spaces from a set of rich geometrical features, and generating motion with a controller using the value function. TRIC can generalize robustly to different situations unseen during training. We tested its performance on a robot grasping task, presented results showing the effect of different training and model settings, and visualized the extracted task space features. We can use the learned model to interpret the most important task dimensions, and their relative weighting in time. A visualization of the hidden layers of the neural network we used for training reveals how TRIC represents a subspace of good grasping poses.

Our method has some limitations. First, it is not suitable for periodic movements and would need to add state information from previous time slices to deal with more complex motions. Second, we assume that an accurate robot kinematic model and sensors for workspace objects' locations are available that provide rich features. Sometimes it is not possible or practical for the teacher to provide such detailed motion information.

6.1 Future Work

An issue for future work is the type of features provided to TRIC as information about the world. Modelling more complex object geometries is important for realistic applications. We can couple TRIC with powerful object representations like implicit surfaces.[5] Additionally, a coupling of perception and manipulation can be achieved in a framework where both objects and motions are represented as potential fields.

It is also possible to use TRIC on the huge amounts of recorded human motion data recently available online, e.g. the CMU Motion Capture Database. The analysis of human demonstrations with TRIC can both teach robots new skills and also uncover interesting results concerning which features underlie human motion generation.

Acknowledgements

This work was supported by the German Research Foundation (DFG), Emmy Noether fellowship TO 409/1-3, and the EU FP7 project TOMSY.

[5] Implicit surface object models are learned from sensory data and the object surface is a nonlinear function potential itself, e.g. a Gaussian Process or SVR, see Steinke et al. (2005).

Appendix

Proof of Proposition 1 for the Direction of IK Generated Motion Steps

Proposition 1 *If $\varrho \to \infty$ then the IK solution q_{t+1} minimizing Equation (9) has the property that the next step $q_{t+1} - q_t$ is approximately proportional to the value function gradient $\mathcal{J}$ in a small region around q_t.*

Proof If $\varrho \to \infty$ then the term $\|f \circ \phi(q) - f \circ \phi(q_t) + \delta\|^2$ of Equation (9) is weighted so high that C_{prior} and any other cost terms we might add are neglected. Let $\mathcal{J}$ be the gradient of the value function $f \circ \phi(q)$ evaluated at $q = q_t$. Using the linearization $f \circ \phi(q_{t+1}) = f \circ \phi(q_t) + \mathcal{J}(q_{t+1} - q_t)$, we can apply the IK Equation (Toussaint (2011)):

$$
\begin{aligned}
q_{t+1} &= q_t - \delta \mathcal{J}^\sharp \\
\mathcal{J}^\sharp &= \left(\varrho \mathcal{J}^T \mathcal{J} + \mathbb{I}\right)^{-1} \mathcal{J}^T \varrho \\
&= \mathcal{J}^T \left(\mathcal{J}\mathcal{J}^T + \varrho^{-1}\right)^{-1} = \frac{1}{\|\mathcal{J}\|^2} \mathcal{J}^T
\end{aligned}
$$

We have used the Woodbury identity and the fact that $\mathcal{J}\mathcal{J}^T = \|\mathcal{J}\|^2$ in the case where we have a 1-dimensional task variable y (in that case the Jacobian is a row vector gradient). $\mathcal{J}^\sharp$ is called the pseudoinverse of $\mathcal{J}$. Thus, the steps generated by our motion model are proportional to $\mathcal{J}$ times a negative scalar number.

Proof of Proposition 2 for Lyapunov Attractor Properties of TRIC

Proposition 2 *Suppose we have trained TRIC on a single trajectory $\{q_t, y_t\}_{t=1}^T$ and that $f \circ \phi(q_T)$ is a minimum of the value function. Additionally, we generate motion with $\varrho \to \infty$, i.e. very high weighting of the value function. Then the motion generated by the model in Equation (8) fulfills the conditions of Theorem 1 and is thus asymptotically stable at the attractor subspace $Q' = \{q' : \phi(q') = \phi(q_T) = y_T\}$.*

Proof Because of $\varrho \to \infty$ the term decreasing the value function f will dominate the motion equation and we can ignore the effect of the other terms. Let's construct $V(q) = f \circ \phi(q) - c_T$, where $c_T = f \circ \phi(q_T)$. Then the Lyapunov stability conditions hold:

- (a) holds directly because of the assumption that $c_T = f \circ \phi(q_T)$ is minimum and any other joint state q s.t. $\phi(q) \neq \phi(q_T)$ will have higher value $f \circ \phi(q)$ than it.
- (b) holds by the construction of $V(q)$ directly implying that

  $$V(q_T) = f \circ \phi(q_T) - c_T = 0$$

- Proposition 1 holds because we assumed that $\varrho \to \infty$. This implies that the steps of the motion model are proportional to the gradient $\mathcal{J}$. Thus the motion model of Equation (9) will make steps decreasing the value $f \circ \phi(q_t)$ constantly and (c) holds.
- (d) holds because $c_T = f \circ \phi(q_T)$ is local minimum and after we reach a joint state q' s.t. $\phi(q') = \phi(q_T)$ the gradient of the value function will be 0 and no further decrease will be possible.

References

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

M. Bain and C. Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129. Oxford University Press, 1996.

M. Berniker and K. Kording. Estimating the sources of motor errors for adaptation and generalization. *Nature Neuroscience*, 11(12):1454–1461, 2008.

A. Billard, Y. Epars, S. Calinon, G. Cheng, and S. Schaal. Discovering optimal imitation strategies. *Robotics and autonomous systems, Special Issue: Robot Learning from Demonstration*, 47(2-3):69–77, 2004.

S. Calinon and A. Billard. Incremental learning of gestures by imitation in a humanoid robot. In *HRI '07: Proceedings of the ACM/IEEE international conference on Human-robot interaction*, pages 255–262, 2007.

J. Call and M. Carpenter. Three sources of information in social learning. *Imitation in animals and artifacts*, pages 211–228, 2002.

J. J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1989. ISBN 0201095289.

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

M. Gienger, M. Toussaint, N. Jetchev, A. Bendig, and C. Goerick. Optimization of fluent approach and grasp motions. In *8th IEEE-RAS International Conference on Humanoid Robots*, 2008.

M. Haindl, P. Somol, D. Ververidis, and C. Kotropoulos. Feature selection based on mutual correlation. In *CIARP*, pages 569–577, 2006.

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

E. S. L. Ho, T. Komura, and C.-L. Tai. Spatial relationship preserving character motion adaptation. *ACM Transactions on Graphics*, 29(4):1–8, 2010.

M. Howard, S. Klanke, M. Gienger, C. Goerick, and S. Vijayakumar. A novel method for learning policies from variable constraint data. *Autonomous Robots*, 27:105–121, 2009.

O. C. Jenkins and M. J. Matarić. A spatio-temporal extension to isomap nonlinear dimension reduction. In *21st Int. Conf. on Machine Learning (ICML)*, 2004.

N. Jetchev. *Learning representations from motion trajectories: analysis and applications to robot planning and control*. PhD thesis, FU Berlin, 2012. URL `http://www.diss.fu-berlin.de/diss/receive/FUDISS\_thesis\_000000037417`. Retrieved on 01/08/12.

N. Jetchev and M. Toussaint. Task space retrieval using inverse feedback control. In *28th Int. Conf. on Machine Learning (ICML)*, pages 449–456, 2011.

S. M. Khansari-Zadeh and A. Billard. Bm: An iterative algorithm to learn stable non-linear dynamical systems with gaussian mixture models. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 2381–2388, 2010.

O. Kroemer, R. Detry, J. H. Piater, and J. Peters. Active learning using mean shift optimization for robot grasping. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 2610–2615, 2009.

O. Kroemer, R. Detry, J. H. Piater, and J. Peters. Grasping with vision descriptors and motor primitives. In *ICINCO (2)*, pages 47–54, 2010.

Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang. A tutorial on energy-based learning. In *Predicting Structured Data*, 2006.

G. Montavon, M. Braun, and K.-R. Müller. Kernel analysis of deep networks. *Journal of Machine Learning Research*, 12:2563–2581, 2011.

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

C. S. Myers and L. R. Rabiner. A comparative study of several dynamic time-warping algorithms for connected word recognition. *The Bell System Technical Journal*, 60(7):1389–1409, 1981.

A. Nouri and M. L. Littman. Dimension reduction and its application to model-based exploration in continuous spaces. *Machine Learning*, 81(1):85–98, 2010.

T. J. Perkins and A. G. Barto. Lyapunov design for safe reinforcement learning. *Journal of Machine Learning Research*, 3:803–832, 2002.

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

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

N. Ratliff, B. Ziebart, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. Srinivasa. Inverse optimal heuristic control for imitation learning. In *Proc. of AISTATS*, pages 424–431, 2009.

N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich. Maximum margin planning. In *26th Int. Conf. on Machine Learning (ICML)*, pages 729–736, 2006.

S. Schaal, J. Peters, J. Nakanishi, and A. J. Ijspeert. Learning movement primitives. In *International Symposium on Robotics Research*, pages 561–572, 2003.

B. Siciliano and O. Khatib, editors. *Springer Handbook of Robotics*. Springer, 2008.

J.-J. Slotine and W. Li. *Applied Nonlinear Control*. Prentice Hall, 1991.

F. Steinke, B. Schölkopf, and V. Blanz. Support vector machines for 3d shape processing. *Computer Graphics Forum*, 24(3):285–294, 2005.

J. Tegin, S. Ekvall, D. Kragic, J. Wikander, and B. Iliev. Demonstration-based learning and control for automatic grasping. *Intelligent Service Robotics*, 2:23–30, 2009.

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

M. Toussaint. Robotics. University Lecture, 2011. URL `http://userpage.fu-berlin.de/~mtoussai/teaching/11-Robotics/`. Retrieved on 01/08/12.

I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

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

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