Trajectory Prediction: Learning to Map Situations to Robot Trajectories

Nikolay Jetchev JETCHEV@CS.TU-BERLIN.DE
Marc Toussaint MTOUSSAI@CS.TU-BERLIN.DE
TU Berlin, Franklinstr. 28/29,10587 Berlin, Germany

Abstract

Trajectory planning and optimization is a fundamental problem in articulated robotics. Algorithms used typically for this problem compute optimal trajectories from scratch in a new situation. In effect, extensive data is accumulated containing situations together with the respective optimized trajectories – but this data is in practice hardly exploited. The aim of this paper is to learn from this data. Given a new situation we want to predict a suitable trajectory which only needs minor refinement by a conventional optimizer. Our approach has two essential ingredients. First, to generalize from previous situations to new ones we need an appropriate situation descriptor – we propose a sparse feature selection approach to find such well-generalizing features of situations. Second, the transfer of previously optimized trajectories to a new situation should not be made in joint angle space – we propose a more efficient task space transfer of old trajectories to new situations. Experiments on a simulated humanoid reaching problem show that we can predict reasonable motion prototypes in new situations for which the refinement is much faster than an optimization from scratch.

1. Introduction

The animal and human ability to generate trajectories quickly is amazing. In typical every-day situations humans do not seem to require time for motion planning but execute complex trajectories instantly. This suggests that there exists a "reactive trajectory policy" which maps "the situation" (or at least motion relevant features of the situation) to the whole trajectory.[1] Such a mapping (if optimal) is utterly complex: the output is not a single current control signal but a whole trajectory which, traditionally, would be the outcome of a computationally expensive trajectory optimization process accounting for collision avoidance, smoothness and other criteria. The input is the current situation, in particular the position of relevant objects for which it is unclear which representation and coordinate systems to use as a descriptor. The goal of this work is to learn such an (approximate) mapping from data of previously optimized trajectories in old situations. We coin this problem *trajectory prediction.*

Existing approaches to trajectory optimization from scratch include traditional gradient-based methods, in particular using a spline-based representation (Zhang & Knoll, 1995), or sequential quadratic programming of which iLQG (Todorov & Li, 2005) is an instance. These methods efficiently converge to local optima which is appropriate when the cost function implies suitable gradients, e.g. to push out of collisions. In more complex situations other methods like Rapidly-exploring Random Trees (RRTs) (Bertram et al., 2006) or probabilistic road maps (Kavraki et al., 1995) are typically used to first find feasible (e.g. collision-free) paths, which are then refined w.r.t. to smoothness and costs using a local optimizer. Such methods are a black-box ingredient to our approach; in the experiment we will mainly utilize iLQG and mention also RRTs.

Concerning our problem of learning from previous optimization data, there exist multiple branches of related work in the literature. In the context of Rein-

Appearing in *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada, 2009. Copyright 2009 by the author(s)/owner(s).

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

forcement Learning the transfer problem has been addressed, where the value function (Konidaris & Barto, 2006) or directly the policy (Stolle & Atkeson, 2007), (Peshkin & de Jong, 2002) is transferred to a new Markov Decision Process. Konidaris and Barto (2006) also discussed the importance of representations for the successful transfer. Although the problem setting is similar, these methods are different in that they do not consider a situation descriptor (or features of the "new" MDP) as an input to a mapping, which directly predicts the new policy or value function.

Related work with respect to exploiting databases of previous trajectories has been proposed in the context of RRTs. Branicky et al. (2008) constructed a compact database of collision free paths that can be reused in future situations to speed up planning under the assumption that some of the previous paths will not be blocked by future obstacles and can be reused for fast planning. Martin et al. (2007) attempted to bias RRTs such that after planning in a set of initial environments, the obstacles can be rearranged and previous knowledge will be used for faster replanning in the new scene; an environment prior, that visits with higher probability states visited in previous trials, is used to speed up planning and use less tree nodes to achieve the final goal. In both cases, the notion of our situation descriptor and the direct mapping to an appropriate new trajectory is missing. Another interesting way to exploit a database of previous motions is to learn a "capability map", i.e., a representation of a robot's workspace that can be reached easily, see (Zacharias et al., 2007). While this allows to decide whether a certain task position can be reached quickly, it does not encode a prediction of a trajectory in our sense.

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. The choice of the situation descriptor is also crucial when trying to generalize data from old to new situations. Our approach is to use feature selection methods to decide on relevant and well-generalizing situation features. To do this, we redundantly blow up the situation descriptor by defining a vector that contains all kinds of distances in various coordinate systems that can be calculated for the current situation – namely a 791-dimensional vector we will define later.

The remainder of the paper is organized as follows. We first provide some background, which sets an appropriate framework to formalize our problem. In section 3 we present our approach to learn a trajectory prediction in new situations, to transfer and refine them. In section 4 we evaluate the approach on some humanoid reaching problems.

2. The Trajectory Prediction Problem

Let $q_t \in \mathbb{R}^n$ be the robot posture as given by all its joint angles at time t. In a given situation x, i.e., for a given initial posture q_0 and the positions of obstacles and targets in this problem instance (we will formally define x below), the problem is to compute a trajectory $\boldsymbol{q} = (q_0, .., q_T)$ for some time horizon T, which fulfils different constraints like reaching a task goal and avoiding collisions. We formulate this as an optimization problem by defining a cost function

$$C(x, \boldsymbol{q}) = \sum_{t=1}^{T} g(q_t) + h(q_t, q_{t\text{-}1}) \ . \tag{1}$$

We will specify such a cost function explicitly in our experiments section. Generally, g will account for task targets and collision avoidance and h for control costs.

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

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

For this we assume to have access to $C(x, \boldsymbol{q})$ and local (linear or quadratic) approximations of $C(x, \boldsymbol{q})$ as provided by a simulator, i.e., we can numerically evaluate $C(x, \boldsymbol{q})$ for given x and $\boldsymbol{q}$ but we have no analytic model.

The problem we address in this paper is to learn an approximate model of the mapping (2) from a data set of previously optimized trajectories. The dataset D comprises pairs of situations and optimized trajectories,

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

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

In this paper we will show the advantages of doing trajectory prediction by adapting basic classification and regression methods to work with pre-calculated values of $C(x, \boldsymbol{q})$ from costly simulations.

3. Prediction and Transfer of a Trajectory to a New Situation

The problem of trajectory prediction as defined in the previous section is generally very hard. We decompose the whole problem into a number of steps which we describe one after another in this section. As an overview, these steps are:

1. Compute the high-dimensional descriptor x for the new situation.
2. Use a learnt mapping to predict a (task space) trajectory for the new situation x.
3. Transfer the trajectory to the new situation using Inverse Kinematics (IK), generating a trajectory $\boldsymbol{q}$.
4. Use a trajectory optimization algorithm to refine $\boldsymbol{q}$, i.e., initialize the algorithm with $\boldsymbol{q}$ and compute the optimal trajectory $\boldsymbol{q}^*$.

We measure the performance of this procedure in terms of how much refinement is needed in the last step (the other steps are computationally cheap). Since the optimizer we use in the refinement (iLQG) is an iterative algorithm, we measure this "refinement cost" as follows

$$F(x, \boldsymbol{q}) = \sum_{j=1}^{J} C(x, \boldsymbol{q}_j) \tag{4}$$

where $\boldsymbol{q}_j$ is the trajectory vector found by the optimizer after j iterations, starting from initialization $\boldsymbol{q}_0 \equiv \boldsymbol{q}$, and J is a constant (we set $J = 40$ as a good compromise between giving a good estimate of the convergence and fast evaluation). This area measure can be interpreted as the area under the optimization curve.

3.1. High-dimensional Situation Descriptor

Step 1 is to compute a high-dimensional situation descriptor. A situation (or problem instance) is fully specified by the initial robot posture q_0 and the positions of obstacles and targets in this problem instance. There are many ways to describe a situation by some vector x. For instance, positions of obstacles could be given relative to some world coordinate system, relative to the robot's base or relative to the endeffector. We should expect that our ability to generalize to new situations crucially depends on how we describe situations. Our approach is to first define a very high-dimentional and redundant situation descriptor which includes distances and relative positions w.r.t. many possible frames of reference. Given this descriptor we use a feature selection technique to infer from the data which of these dimensions are best for trajectory prediction in new situations.

Concretely, in our reaching task experiments, $x \in \mathbb{R}^s$ is defined as a s=791-dimensional vector. We have 31 robot joint angles in the posture vector q_0. We have 20 objects for which we measure pairwise distances and rotations, 18 body parts (upper body of our robot), the obstacle object (a single table in our scenario) and the reach target location. This makes 190 combinations of such object pairs. For each pair i we measure the 3D relative distance $p_i = (p_i^x, p_i^y, p_i^z)$ of the object centers and the 1D cosine o_i of the z axis relative orientation of the objects.

$$(q_0, p_1, ..., p_{190}, o_1, ..., o_{190}) \in \mathbb{R}^{791} \tag{5}$$

Even larger spaces are possible, for example taking polar coordinates and relative object coordinate frames instead of the world frame, but the choice of (5) turned out to be sufficient in our experiments.

3.2. Trajectory Prediction

As in typical kernel machines, at the core of a good predictor is a good choice of similarity measure (kernel) in input space, see (Scholkopf & Smola, 2001). We consider rather basic prediction methods for (2)– namely nearest neighbour (NN) and locally weighted regression (LWR) – but spend some effort in training a suitable similarity measure in situation space. Given the data set $D = \{(x_i, \boldsymbol{q}_i)_{i=1}^d\}$ and a new situation x^*, we want to compute a similarity between x^* and each x_i in the data set,

$$k(x^*, x_i) = \exp\{-\frac{1}{2}(x^* - x_i)^\top W(x^* - x_i)\} \tag{6}$$

$$W = \text{diag}(w_1^2, .., w_s^2)\ , \tag{7}$$

where the entries w_j^2 of the diagonal metric parametrize the weighting of the jth dimension in the situation descriptor. Given this metric, the NN predictor is

$$f(x) = \mathcal{T}_{x_i x} \boldsymbol{q}_{\hat{i}} \ , \quad \hat{i} = \operatorname*{argmax}_i k(x_i, x) \ , \tag{8}$$

where $\mathcal{T}_{x_i x} = \phi_x^{-1} \circ \phi_{x_i}$ is a transfer operator which first projects to a task space in situation x_i and then projects back to a joint trajectory in situation x – we will explain this operator in detail in the next section. For K nearest neighbours, the LWR predictor is

$$f(x) = \phi_x^{-1} \ \frac{\sum_i^K k(x, x_i) \ \phi_{x_i} \boldsymbol{q}_i}{\sum_j^K k(x, x_j)} \ . \tag{9}$$

Note that for $K = 1$ the two methods are identical. For higher values of K the similarity $k(x, x_i)$ controls how we average between the nearest neighbours in task space and we remain always in the convex hull of these neighbours, which assures meaningful trajectory outputs. We also tried Kernel Ridge Regression (KRR) but observed worse performance, in particular when the new situation is far from all previously seen (all $k(x, x_i)$ are small) and KRR essentially returns the global mean of all trajectories – this global interpolation typically seemed not useful.

We train the parameters $w = (w_1, ..., w_s)$ on the basis of the performance measure $F(x, \boldsymbol{q})$ defined in (4). Assume we had a probabilistic model $f(x)$ that only outputs (transferred) trajectories $\boldsymbol{q}_i$ from the database and selects them with probability

$$P(f(x) = \mathcal{T}_{x_i x} \boldsymbol{q}_i) = \frac{1}{Z} \exp\{-\frac{1}{2}(x - x_i)^\top W (x - x_i)\} \tag{10}$$

where $Z = \sum_i \exp\{-\frac{1}{2}(x - x_i)^\top W (x - x_i)\}$ ensures normalization. The expected cost of this probabilistic mapping over D is

$$\mathrm{E}\{F(x, f(x))\} = \sum_{i=1}^{d} P(f(x) = \mathcal{T}_{x_i x} \boldsymbol{q}_i) F(x, \mathcal{T}_{x_i x} \boldsymbol{q}_i) \ . \tag{11}$$

We choose weights w so that $\mathrm{E}\{F(x, f(x))\}$ is minimal for all x. Our approach has some analogies with other kernel training and feature selection methods, see (Lowe, 1995).The derivative of this expected cost, with respect to the vector w can be calculated as:

$$\frac{\partial \mathrm{E}\{F(x, f(x))\}}{\partial w} = \sum_{i=1}^{d} F(x, f(x)) \frac{\partial P(f(x) = \mathcal{T}_{x_i x} \boldsymbol{q}_i)}{\partial w} \tag{12}$$

$$\frac{\partial P(f(x) = \mathcal{T}_{x_i x} \boldsymbol{q}_i)}{\partial w} = \frac{Z \frac{\partial \tilde{P}_i}{\partial w} - P(f(x) = \mathcal{T}_{x_i x} \boldsymbol{q}_i) \sum_{j=1}^{d} \frac{\partial \tilde{P}_j}{\partial w}}{Z^2} \tag{13}$$

$$\frac{\partial \tilde{P}_i}{\partial w} = \frac{\partial \exp\{-\frac{1}{2}(x - x_i)^\top W (x - x_i)\}}{\partial w} = -\tilde{P}_i (x - x_i)^2 diag(w) \tag{14}$$

where $\tilde{P}_i = \exp\{-\frac{1}{2}(x - x_i)^\top W (x - x_i)\}$ is the unnormalized probability. Given a second dataset $\bar{D}$ of situations for which we measured all costs F of situations from D, we want to minimize:

$$\sum_{x \in \bar{D}} \mathrm{E}\{F(x, f(x))\} + \lambda |w|_1 \tag{15}$$

where λ controls the trade-off with the regularization. Having the squares of w on the diagonal of W ensures that we get a positive matrix W with an unconstrained minimization algorithm. We used the L_1 norm of w to get a sparse solution and took $sign(w)$ as its gradient, see (Schmidt et al., 2007).

3.3. Task Space Transfer

In the previous sections we already used the transfer operator $\mathcal{T}_{x_i x}$ that transfers a data trajectory from situation x_i to a new situation x. We propose to do this transfer in task space. Assume we have a kinematic mapping $\phi_x : q_t \mapsto y_t$ which maps a joint configuration $q_t \in \mathbb{R}^n$ to a task vector $y_t \in \mathbb{R}^m$, for instance the endeffector position of the robot. Generally, such a mapping depends on the situation x, e.g. the positions of other objects. Then we can project a joint space trajectory $\boldsymbol{q}$ into the task space; we write $\boldsymbol{y} = \phi_x(\boldsymbol{q}) \in \mathbb{R}^{T+1 \times m}$ for this, meaning that ϕ is applied in every time slice. In a new situation (where we know the initial posture q_0), this task space trajectory can be projected back to joint space using inverse kinematics (IK), see (Nakamura & Hanafusa, 1987). Stretching rigorous mathematical notation a bit, we write this IK projection of a task trajectory as $\phi_x^{-1}(\boldsymbol{y})$. In this notation, the transfer operator is the concatenation $\mathcal{T}_{x' x} = \phi_x^{-1} \circ \phi_{x'}$.

Generally, the task space can be freely defined in this approach. In our experiments we chose the task space to be the 3D endeffector position relative to an obstacle (table) coordinate system. This choice of task space helps to generalize to translations in obstacle position, see (Berniker & Kording, 2008). We will explain more details on the used inverse kinematics in the experimental section.

3.4. A Clustering Approach to Prediction

As an alternative to the above prediction scheme and for empirical evaluation we also test a simple clustering approach where we choose from a small predefined movement set:

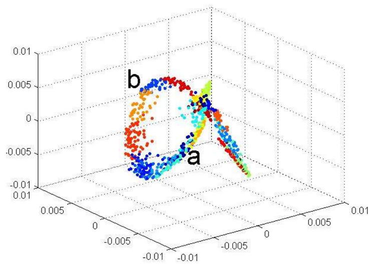
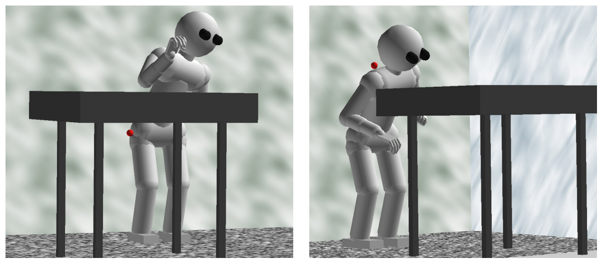


Figure 1. Low dimensional embedding of D using Euclidean distance matrix between $\phi_{x_i}(\boldsymbol{q}_i)$. Colours indicate assigned cluster. The situations x_a and x_b are visualized in Figure 2



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

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

1. Cluster the task space trajectories $\phi_{x_i}(\boldsymbol{q}_i)$ using standard Euclidean distance in c clusters with centroids $\{\boldsymbol{y}_i\}_{i=1}^c$.

2. Gather training data pairs (x, i^*) where $i^* = \text{argmin}_{i \in C} F(x, \phi_x^{-1}\boldsymbol{y}_i)$.

3. Train a supervised classification algorithm on this data, so we can predict $f(x) = \phi_x^{-1}\boldsymbol{y}_{i^*}$.

We were motivated for this approach by the good low dimensional structure we found in D, see Figure 1, where different regions correspond to characteristic movements and situations as in Figure 2. More elaborate ways to code generalized movements and primitives have been proposed, like Hidden Markov Models (Calinon & Billard, 2005; Shon et al., 2007), but cluster analysis has the advantage of simplicity. The task space trajectories $\boldsymbol{y}_i$ correspond to prototype avoidance paths around the obstacle in Figure 3, since mapping ϕ_{x_i} projects successfully optimized trajectories in relative endeffector-table coordinates.

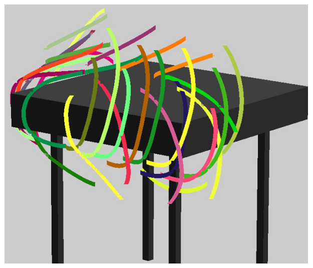


Figure 3. Prototype trajectories $\boldsymbol{y}_i$, tracing good avoidance paths around the obstacle.

4. Experiments

4.1. Problem setup

The scenario we examined contains a humanoid robot body with the right hand index finger as endeffector, a reaching target (red point) and an obstacle (the table) as seen in Figure 2. The task for the robot is to reach the target with the endeffector without colliding with the obstacle. Different scenarios are generated by uniformly sampling the position of the table in a rectangle of size (0.9, 02, 0.2), the target in (0.5, 0.2, 0.6), and initial endeffector position in (0.3, 0.3, 0.9). Situations with initial collisions and too easy situations where the endeffector was closer than 0.3 to the target were discarded in order to avoid trivial situations and to put a greater focus on more challenging scenarios, where the endeffector must move on the other side of the table to reach the target. With this generative model we gathered a database D with 1000 situations for use in our experiments. We set $T = 200$ to allow for a smooth and detailed movement description.

We chose the term h in the cost function (1) to enforce a trajectory of short length with smooth transitions between the trajectory steps. We define h as

$$h(q_t, q_{t\text{-}1}) = (q_t - q_{t\text{-}1})^2 \ . \tag{16}$$

The cost term g in (1) is defined as

$$g(q_t) = g_1(q_t) + g_2(q_t) \tag{17}$$

where g_1 penalizes collisions while executing the grasp movement. The value of this collision cost is the sum of the pairwise penetration depths c_i of colliding objects. Minimizing it moves the robot body parts away from obstacles.

$$g_1(q_t) = 10^5 \sum_i c_i^2 \tag{18}$$

The task of reaching the target position with the endeffector is represented in g_2. We want the target to be reached at the end of the movement, so we define this cost function to have a higher value for $t = T$:

$$g_2(q_t) = \begin{cases} d^2 & t < T \\ 10^4 d^2 & t = T \end{cases} \tag{19}$$

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

For the mapping ϕ^{-1} we used the prioritized iterated IK method similar to (Baerlocher & Boulic, 2004), which allows to include additional task constraints with priorities. We have three such constraints: to follow the task space trajectory $\phi(\boldsymbol{q})$, to avoid collisions as in (18), and to reach the target in the final time step as in (19). We imposed higher priority on the collision avoidance, that is, the task space transfer tries to follow the given task space trajectory roughly while avoiding any collisions and correcting for them.

4.2. Methods Compared

We evaluated several methods for trajectory prediction:

- *Linear* stands for the baseline method usually used for initialization of motion planners. It predicts a straight line trajectory of the endeffector to the target, smoothed by IK (with same constraints as in the previous section).
- NN^{Euclid} predicts a trajectory using (8) with the default Euclid metric.
- LWR^{Euclid} uses for prediction (9) with 3 neighbours and Euclid metric.
- NN^{Opt} uses NN selection (8) with the metric optimized in (15).
- LWR^{Opt} uses (9) with 3 neighbours and trained metric.
- *Cluster* uses initialization with the clustered prototypes C as in section 3.4.

In the case of NN^{Opt} we used cross-validation to choose the regularization $\lambda = 0.03$, scaled all features to $[-1, 1]$, optimized for about 20 Quasi-Newton BFGS iterations and got a sparse metric with 52 non-zero coefficients w_i^2. Figure 5 shows the decay of the feature coefficients and Table 1 highlights the top features. Most of these encode highly collision or target relevant information. The descriptor is compressed from 791 to 52 non-zero features.

In the case of *Cluster* we used spectral clustering of D and chose the number of clusters to be 30. We made a new set of 900 situations $\hat{D}$, and for each $x \in \hat{D}$ measured $F(x, \phi^{-1}(\hat{\boldsymbol{y}}_i))$ for all $\hat{\boldsymbol{y}}_i \in C$, for a total of 27000 evaluations. We had 75% accuracy in predictions with Linear SVM, and the bad classifications were usually with a cluster that also has quite good costs.

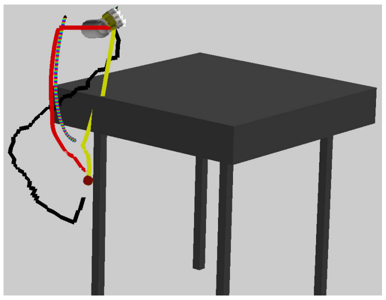


Figure 4. Predicted trajectory of the endeffector after task space transfer. Yellow corresponds to *Linear*, red is *Cluster* with $\boldsymbol{y}$ the striped line, black is from a RRT.

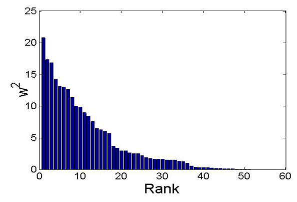


Figure 5. Features ordered by weight w_i^2

Figure 4 gives a brief illustration of the *Cluster* method. For comparison we also included an RRT method (with 2000 nodes), which was worse than *Linear* for our problem setup but might be more competitive in difficult cluttered environments.

4.3. Results

We measure total computation time (initialization and iLQG) as follows: for a generated test set of 1000 situations (different from D), run all initialization algorithms that are compared and measure the cost along 100 iLQG iterations (each of which lasts 0.065s). Let $C_{\min}(x)$ be the minimal achieved cost of all methods for situation x. Compute the computation time for each of the algorithms as the time in seconds until $C_{\min}(x) + \epsilon$ is reached. A margin $\epsilon = 0.2$ implies a feasible solution without collisions whereas a smaller margin like $\epsilon = 0.05$ corresponds to solutions which are near the optimum.

In Table 2, # stands for the number of 1000 test situations that did not reach level ϵ using the particular method. μ is the average time to reach level ϵ, calculated on the 773 situations where *all* of the levels for all methods were reached. With ± the Standard Mean

Table 1. Several highly ranked features.

Feature	w_i^2
$p^y_{waist-table}$	20.7
$p^x_{chest-table}$	17.3
$p^x_{target-table}$	16.8
$p^y_{back-table}$	14.2
$p^z_{neck-target}$	11.4
$p^z_{wristR-target}$	8.9

Error of our estimate of μ is shown.

In Figure 6 we show performance after 0.75s (initialization and 10 iLQG iterations) averaging for 1000 situations the distance ϵ from the optimal cost $C_{\min}(x)$ for the current iteration. We make the following observations from Table 2 and Figure 6:

- All of the 5 prediction methods using database D improved greatly the convergence speed.The predicted trajectories take the endeffector on a collision free path around the table before any iLQG refinement, unlike *Linear* which is often stuck in collisions at first, see Figure 4. The predictions also allowed iLQG to reach better optima much more often, as seen in the # values in Table 2.

- The trained similarity metric improves significantly NN^{Opt} over NN^{Euclid} in all statistics from Figure 6 and Table 2.

- The *Cluster* and NN^{Opt} methods are the best; there is not a clear winner between them. Eventually, the choice will be influenced by additional criteria like time to train and memory.

  Cluster has the least number of failures # from all methods. A possible explanation is that the clusters C robustly represent good control trajectories for all situations. Combined with the good classification accuracy of the SVM, this allows for minima even in the few situations where NN^{Opt} initialization fails.

  On the other hand, *Cluster* has worse time μ than NN^{Opt} and has a larger area under the curve in Figure 6, especially in the first iterations.

- Methods LWR^{Euclid} and LWR^{Opt} were slightly worse compared to the respective NN methods. A possible reason is that LWR is more sensitive to parameters, so more data for training of the situation metric can improve it.

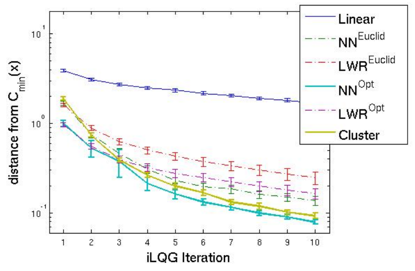


Figure 6. Convergence averaged over 1000 situations.

5. Conclusion

In this paper we addressed the problem of trajectory prediction: we want to exploit data from previous trajectory optimizations to predict reasonable trajectories in new situations. We proposed two key aspects to solve this problem: an appropriate situation descriptor and a task space transfer of previously optimized trajectories to new situations. Concerning the situation descriptor, we demonstrated that learning a (L_1-regularized) metric in a high-dimensional descriptor significantly increases performance of the mapping. Interestingly, this means that we can extract features of a situation (e.g., choose from a multitude of possible coordinate systems) that generalize well w.r.t. trajectory prediction. As an alternative a clustering approach yielded equally good empirical results. The task space transfer is motivated by the fact that a naive replay of a trajectory in a new situation will not yield good motions, in particular w.r.t. collisions. The task space transfer – that is, first projecting an old trajectory to a task space and then projecting it back, in the new situation, using IK – allows an adaptation to the new situation implicit in the inverse kinematics.

One limitation of our current approach is the assumption of perfect information and simple object geometry. This can be addressed by sensor-driven situation descriptors, but these will also increase the problem dimensionality and require algorithms that scale well. Future research will focus on more cluttered scenes and more difficult tasks, requiring different cost functions and situation descriptors. Collecting a database D will be more costly for such tasks, but then trajectory prediction has the potential to save more time in the online phase by finding and predicting complicated movement patterns between different situations.

Acknowledgments

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

Table 2. Performance of different initialization methods. μ is average time in seconds, # is count of failures.

Method		$\epsilon = 0.2$	$\epsilon = 0.15$	$\epsilon = 0.1$	$\epsilon = 0.05$
Linear	#	31	41	63	155
	μ	1.26 ± 0.04	1.29± 0.05	1.4 ± 0.05	1.96 ± 0.07
NN^{Euclid}	#	4	6	9	52
	μ	0.4± 0.01	0.47 ± 0.01	0.62 ± 0.02	1.14 ± 0.04
LWR^{Euclid}	#	4	9	19	58
	μ	0.39 ± 0.01	0.45 ± 0.01	0.59 ± 0.02	1.05 ± 0.04
NN^{Opt}	#	2	3	4	33
	μ	**0.32** ± 0.01	**0.36** ± 0.01	0.47 ± 0.01	0.83 ± 0.02
LWR^{Opt}	#	3	4	16	49
	μ	**0.31** ± 0.01	**0.35** ± 0.01	**0.45** ± 0.01	**0.76** ± 0.03
Cluster	#	**1**	**1**	**2**	**16**
	μ	0.39 ± 0.01	0.45 ± 0.01	0.57 ± 0.01	1.02 ± 0.03

References

Baerlocher, P., & Boulic, R. (2004). An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. *The Visual Computer: Int. Journ. of Computer Graphics*, *20*, 402–417.

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

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

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

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

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

Kavraki, L. E., Latombe, J.-C., Motwani, R., & Raghavan, P. (1995). Randomized query processing in robot path planning. *Twenty-seventh annual ACM Symposium on Theory of Computing (STOC)* (pp. 353–362).

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

Lowe, D. G. (1995). Similarity metric learning for a variable-kernel classifier. *Neural Computation*, *7*, 72–85.

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

Nakamura, Y., & Hanafusa, H. (1987). Optimal redundancy control of robot manipulators. *Int. Journ. of Robotic Research*, *6*, 32–42.

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

Schmidt, M., Fung, G., & Rosales, R. (2007). Fast optimization methods for l1 regularization: A comparative study and two new approaches. *18th European Conf. on Machine Learning (ECML)* (pp. 286–297).

Scholkopf, B., & Smola, A. J. (2001). *Learning with kernels: Support vector machines, regularization, optimization, and beyond.* Cambridge, MA, USA: MIT Press.

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

Stolle, M., & Atkeson, C. (2007). Knowledge transfer using local features. *IEEE Int. Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL).* (pp. 26–31).

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

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

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

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