Learning Trajectory Preferences for Manipulators via Iterative Improvement

Ashesh Jain ASHESH@CS.CORNELL.EDU
Thorsten Joachims TJ@CS.CORNELL.EDU
Ashutosh Saxena ASAXENA@CS.CORNELL.EDU
Department of Computer Science, Cornell University

Abstract

We consider the problem of learning good trajectories for manipulation tasks. This is challenging because the criterion defining a good trajectory varies with users, tasks and environments. In this paper, we propose a co-active online learning framework for teaching robots the preferences of its users for object manipulation tasks. The key novelty of our approach lies in the type of feedback expected from the user: the human user does not need to demonstrate optimal trajectories as training data, but merely needs to iteratively provide trajectories that slightly improve over the trajectory currently proposed by the system. We argue that this co-active preference feedback can be more easily elicited from the user than demonstrations of optimal trajectories, while, nevertheless, theoretical regret bounds of our algorithm match the asymptotic rates of optimal trajectory algorithms. We demonstrate the generalization ability of our algorithm on a variety of tasks, for whom, the preferences were not only influenced by the object being manipulated but also by the surrounding environment.

1. Introduction

Mobile manipulator robots have arms with high degrees of freedom (DoF), enabling them to perform household chores (e.g., PR2) or complex assembly-line tasks (e.g., Baxter). In performing these tasks, a key problem lies in identifying an appropriate trajectory. An appropriate trajectory not only needs to be valid from a geometric point (i.e., feasible and obstacle-free, the criterion that most planners focus on), but it also needs to satisfy the user's preferences.

Such user's preferences over trajectories vary between users, between tasks, and between the environments the trajectory is performed in. For example, a glass of water should be moved in an upright position without jerks while maintaining a safe distance from nearby electronic items, trajectories of sharp objects should be kept a safe distance from humans, etc. These preferences are often hard to describe and anticipate without knowing where and how the robot is deployed. This makes it infeasible to manually encode (e.g. [19]) them in existing path planners a priori [7, 26, 31].

In this work we propose an algorithm for learning user preferences over trajectories through interactive feedback from the user in a co-active learning setting [30]. Unlike in other learning settings, where a human first demonstrates optimal trajectories [3] for a task to the robot, our learning model does not rely on the users ability to demonstrate optimal trajectories a priori. Instead, our learning algorithm explicitly guides the learning process and merely requires the user to incrementally improve the robots trajectories. From these interactive improvements the robot learns a general model of the user's preferences in an online fashion. We show empirically that a small number of such interactions is sufficient to adapt a robot to a changed task. *Since the user does not have to demonstrate a (near) optimal trajectory to the robot*, we argue that our feedback is easier to provide and more widely applicable. Nevertheless, we will show that it leads to an online learning algorithm with provable regret bounds that decay at the same rate as if optimal demonstrations were available.

In our empirical evaluation, we learn preferences for a high DoF personal robot (PR2) performing a variety of household chores in simulation. By designing expressive trajectory features, we show how our algorithm learn preferences from online user feedback on a broad range of tasks for which object properties are of particular importance (e.g., manipulating sharp objects with humans in vicinity). To the best of our knowledge, no previous work has learned such fine-grained preferences over trajectories. We extensively evaluate our approach on a set of 35 household tasks, both in batch experiments as well as through a user study. Our results show that our system not only quickly learns good trajectories on individual tasks, but also generalizes well to tasks that the algorithm has not seen before.

Proceedings of the 30^{th} International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

2. Related Work

Teaching a robot to produce desired motions has been a long standing goal and several approaches have been studied. Most of the past research has focussed on mimicking expert's demonstrations, for example, autonomous helicopter flights [1], ball-in-a-cup experiment [18], planning 2-D paths [25, 27, 28], etc. Such a setting (learning from demonstration, LfD) is applicable to scenarios when it is clear to an expert what constitutes a good trajectory. In many scenarios, especially involving high DoF manipulators, this is extremely challenging to do.[1] This is because the users have to give not only the end-effector's location at each time-step, but also the full configuration of the arm in a way that is spatially and temporally consistent. In our setting, the user never discloses the optimal trajectory (or provide optimal feedback) to the robot, but instead, the robot learns preferences from suboptimal suggestions for how the trajectory can be improved.

Some later works in LfD provided ways for handling noisy demonstrations, under the assumption that demonstrations are either near optimal [36] or locally optimal [22]. Providing noisy demonstrations is different from providing relative preferences, which are biased and can be far from optimal. We compare with an algorithm for noisy LfD learning in our experiments. A recent work [34] leverages user feedback to learn rewards of a Markov decision process. Since they learn in discretized action space their approach is not directly applicable to manipulators with large action spaces.

Our application scenario of learning trajectories for high DoF manipulations performing tasks in presence of different objects and environmental constraints goes beyond the application scenarios that previous works have considered. In such cases, modeling them results in a large state space and makes most state-of-the-art LfD approaches intractable. In our work, we design appropriate features that consider robot configurations, object-object relations, and temporal behavior, and use them to learn a score function representing the preferences in trajectories.

In other related works, Berenson et al. [4] and Phillips et al. [24] consider the problem of trajectories for high-dimensional manipulators. They store prior trajectories for computational reasons for different tasks. These methods are complementary to ours, in that we could leverage their database of trajectories and train our system on samples drawn from it. Other recent works such as [10, 32] consider generating human-like trajectories. These works are complementary to ours in that humans-robot interaction is an important aspect and such ideas could be incorporated in our approach.

[1] Consider the following analogy. In search engine results, it is much harder for the user to provide the best web-pages for each query, but it is easier to provide relative ranking on the search results by clicking.

3. Learning and Feedback Model

We model the learning problem in the following way. For a given task, the robot is given a context x that describes the environment, the objects, and any other input relevant to the problem. The robot has to figure out what is a good trajectory y for this context. Formally, we assume that the user has a scoring function $s^*(x, y)$ that reflects how much he values each trajectory y for context x. The higher the score, the better the trajectory. Note that this scoring function cannot be observed directly, nor do we assume that the user can actually provide cardinal valuations according to this function. Instead, we merely assume that the user can provide us with *preferences* that reflect this scoring function. The robots goal is to learn a function $s(x, y; w)$ (where w are the parameters to be learned) that approximates the users true scoring function $s^*(x, y)$ as closely as possible.

Interaction Model. The learning process proceeds through the following repeated cycle of interactions.

Step 1: The robot receives a context x. It then uses a planner to sample a set of trajectories, and ranks them according to its current approximate scoring function $s(x, y; w)$.
Step 2: The user either lets the robot execute the top-ranked trajectory, or corrects the robot by providing an improved trajectory $\bar{y}$. This provides feedback indicating that $s^*(x, \bar{y}) > s^*(x, y)$.
Step 3: The robot now updates the parameter w of $s(x, y; w)$ based on this preference feedback and returns to step 1.

Regret. The robot's performance will be measured in terms of regret, $REG_T = \frac{1}{T}\sum_{t=1}^{T}[s^*(x_t, y_t^*) - s^*(x_t, y_t)]$, which compares the robot's trajectory y_t at each time step t against the optimal trajectory y_t^* maximizing the user's unknown scoring function $s^*(x, y)$, $y_t^* = argmax_y s^*(x_t, y)$. Note that the regret is expressed in terms of the user's true scoring function s^*, even though this function is *never observed*. Regret characterizes the performance of the robot over its whole lifetime, therefore reflecting how well it performs *throughout* the learning process. As we will show in the following sections, we will employ learning algorithms with theoretical bounds on the regret for scoring functions that are linear in their parameters, making only minimal assumptions about the difference in score between $s^*(x, \bar{y})$ and $s^*(x, y)$ in Step 2 of the learning process.

User Feedback and Trajectory Visualization. Since the ability to easily give preference feedback in Step 2 is crucial for making the robot learning system easy to use for humans, we designed two interfaces that enable the user to easily provide improved trajectories.

(a) Re-ranking: We display the ranking of trajectories using OpenRave [9] and ask the user to identify whether a lower-ranked trajectory is better than the top-ranked one. Section 5.1 studies this feedback in detail.
(b) Interactive: We built an interactive Rviz-ROS [12] interface that allows the user to improve a trajectory by cor-

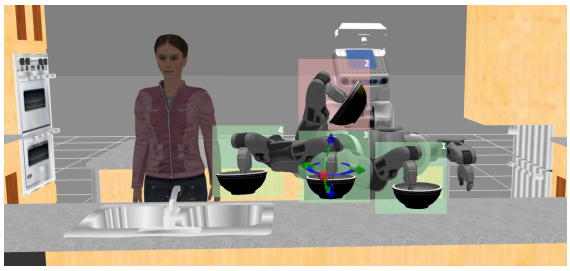


Figure 1: **Our user feedback system.** Different trajectories are shown to users to select from. In this example, interactive feedback allows a user to move waypoint 2 (in red) to waypoint 3 (in green). Waypoint 3 shows interactive markers that guide the user.

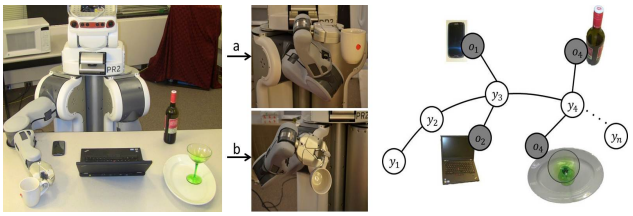


Figure 2: **(Left)** An environment with a few objects where the robot was asked to move the cup on the left to the right. **(Middle)** There are two ways of moving it, 'a' and 'b', both are suboptimal in that the arm is contorted in 'a' but it tilts the cup in 'b'. Given such constrained scenarios, we need to reason about such subtle preferences. **(Right)** We encode preferences concerned with object-object interactions in a score function expressed over a graph. Here $y_1, \ldots, y_n$ are different waypoints in a trajectory. The shaded nodes corresponds to environment (table node not shown here). Edges denotes interaction between nodes.

recting one of its waypoints. Figure 1 shows a robot moving a bowl with one bad waypoint (in red), and the user provides a feedback by correcting it. This feedback is useful (i) for bootstrapping the robot, (ii) for avoiding local maxima where the top trajectories in the ranked list are all bad but ordered correctly, and (iii) when the user is satisfied with the top ranked trajectory except for minor errors.

Note that in both re-ranking and interactive feedback, the user never reveals the optimal trajectory to the algorithm but just provides a slightly improved trajectory.

4. Learning Algorithm

For each task, we model the user's scoring function $s^*(x, y)$ with the following parameterized family of functions.

$$s(x, y; w) = w \cdot \phi(x, y) \tag{1}$$

w is a weight vector that needs to be learned, and $\phi(\cdot)$ are features describing trajectory y for context x. We further decompose the score function in two parts, one only concerned with the objects the trajectory is interacting with, and the other with the object being manipulated and the environment

$$\begin{aligned} s(x, y; w_O, w_E) &= s_O(x, y; w_O) + s_E(x, y; w_E) \\ &= w_O \cdot \phi_O(x, y) + w_E \cdot \phi_E(x, y) \end{aligned} \tag{2}$$

We now describe the features for the two terms, $\phi_O(\cdot)$ and $\phi_E(\cdot)$ in the following.

4.1. Features Describing Object-Object Interactions

This feature captures the interaction between objects in the environment with the object being manipulated. We enumerate waypoints of trajectory y as $y_1, .., y_N$ and objects in the environment as $\mathcal{O} = \{o_1, .., o_K\}$. The robot manipulates the object $\bar{o} \in \mathcal{O}$.

A few of the trajectory waypoints would be affected by the other objects in the environment. For example in Figure 2, o_1 and o_2 affect the waypoint y_3 because of proximity. Specifically, we connect an object o_k to a trajectory waypoint if the minimum distance to collision is less than a threshold or if o_k lies below $\bar{o}$. The edge connecting y_j and o_k is denoted as $(y_j, o_k) \in \mathcal{E}$.

Since it is the attributes [20] of the object that really matter in determining the trajectory quality, we represent each object with its *attributes*. Specifically, for every object o_k, we consider a vector of M binary variables $[l_k^1, .., l_k^M]$, with each $l_k^m = \{0, 1\}$ indicating whether object o_k possesses property m or not. For example, if the set of possible properties are {heavy, fragile, sharp, hot, liquid, electronic}, then a laptop and a glass table can have labels $[0, 1, 0, 0, 0, 1]$ and $[0, 1, 0, 0, 0, 0]$ respectively. The binary variables l_k^p and l^q indicates whether o_k and $\bar{o}$ possess property p and q respectively.[2] Then, for every (y_j, o_k) edge, we extract following four features $\phi_{oo}(y_j, o_k)$: projection of minimum distance to collision along x, y and z (vertical) axis and a binary variable, that is 1, if o_k lies vertically below $\bar{o}$, 0 otherwise. We now define the score $s_O(\cdot)$ over this graph as follows:

$$s_O(x, y; w_O) = \sum_{(y_j, o_k) \in \mathcal{E}} \sum_{p,q=1}^{M} l_k^p l^q [w_{pq} \cdot \phi_{oo}(y_j, o_k)]$$

Here, the weight vector w_{pq} captures interaction between objects with properties p and q. We obtain w_O in eq. (2) by concatenating vectors w_{pq}. More formally, if the vector at position i of w_O is w_{uv} then the vector corresponding to position i of $\phi_O(x, y)$ will be $\sum_{(y_j, o_k) \in \mathcal{E}} l_k^u l^v [\phi_{oo}(y_j, o_k)]$.

4.2. Trajectory Features

We now describe features, $\phi_E(x, y)$, obtained by performing operations on a set of waypoints. They comprise the following three types of the features:

[2] We assume that the objects attributes have been extracted using an algorithm such as in [20].

Robot Arm Configurations. While a robot can reach the same operational space configuration for its wrist with different configurations of the arm, not all of them are preferred [35]. For example, the contorted way of holding the mug shown in Figure 2 may be fine at that time instant, but would present problems if our goal is to perform an activity with it. We compute features capturing robot's arm configuration using the location of its elbow and wrist, w.r.t. to its shoulder, in cylindrical coordinate system, (r, θ, z). We divide a trajectory into three parts in time and compute 9 features for each of the parts. These features encode the maximum and minimum r, θ and z values for wrist and elbow in that part of the trajectory, giving us 6 features. Since at the limits of the manipulator configuration, joint locks may happen, therefore we also add 3 features for the location of robot's elbow whenever the end-effector attains its maximum r, θ and z values respectively. Therefore features obtained from robot arm configuration $\phi_{robot}(\cdot) \in \mathbb{R}^9$ (3+3+3=9) for each one-third part and $\phi_{robot}(\cdot) \in \mathbb{R}^{27}$ for the complete trajectory.

Orientation and Temporal Behavior of the Object to be Manipulated. Object orientation during the trajectory is crucial in deciding its quality. For some tasks, the orientation must be strictly maintained (e.g., moving a cup full of coffee); for others, it needs to be maintained within a limit (e.g., moving a bowl partially full of fruits); and for some others, it may be necessary to change it in a particular fashion (e.g., in pouring activity). Different parts of the trajectory may have different requirements over time. For example, in the placing task, we may need to bring the object closer to obstacles and be more careful. We therefore divide trajectory into three parts in time. For each part we store the cosine of the object's maximum deviation, along the vertical axis, from its final orientation at the goal location. To capture tremors or oscillations experienced by the object during trajectory, we obtain a spectrogram for each one-third part for the movement of the object in x, y, z directions as well as for the deviation along vertical axis. We then compute the average power spectral density (PSD) in the low frequency and the high-frequency part as six additional features for each. This gives us 9 (=1+4*2) features for each one-third part. Together with one additional feature of object's maximum deviation along the whole trajectory, we get $\phi_{obj}(\cdot) \in \mathbb{R}^{28}$ (=9*3+1).

Object-Environment Interactions. This feature captures temporal variation of vertical and horizontal distances of the object $\bar{o}$ from its surrounding surfaces. In detail, we divide the trajectory into three equal parts, and for each part we compute object's: (i) minimum vertical distance from the nearest surface below it. (ii) minimum horizontal distance from the surrounding surfaces; and (iii) minimum distance from the table, on which the task is being performed, and (iv) minimum distance from the goal location. We also take an average, over all the waypoints, of the horizontal and vertical distances between the object and the nearest surfaces around it. To capture temporal variation of object's distance from its surrounding we plot a time-frequency spectrogram of the object's vertical distance from the nearest surface below it, from which we extract six features by dividing it into grids. This feature is expressive enough to differentiate whether an object just grazes over table's edge (steep change in vertical distance) versus, it first goes up and over the table and then moves down (relatively smoother change). Thus, the features obtained from object-environment interaction are $\phi_{obj-env}(\cdot) \in \mathbb{R}^{20}$ (3*4+2+6=20).

Final feature vector is obtained by concatenating $\phi_{obj-env}$, ϕ_{obj} and ϕ_{robot}, giving us $\phi_E(\cdot) \in \mathbb{R}^{75}$.

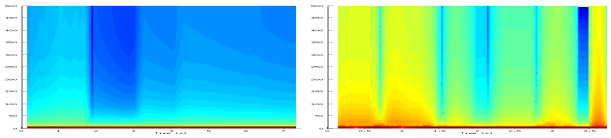


Figure 3: Spectrogram for movement in z-direction. (Left) Good trajectory. (Right) A bad trajectory.

4.3. Computing Trajectory Rankings

For obtaining the top trajectory (or a top few) for a given task with context x, we would like to maximize the current scoring function $s(x, y; w_O, w_E)$.

$$y^* = \arg\max_y s(x, y; w_O, w_E). \tag{3}$$

Note that this poses two challenges. First, trajectory space is continuous and needs to be discretized to maintain argmax in (3) tractable. Second, for a given set $\{y^{(1)}, \ldots, y^{(n)}\}$ of discrete trajectories, we need to compute (3). Fortunately, the latter problem is easy to solve and simply amounts to sorting the trajectories by their trajectory scores $s(x, y^{(i)}; w_O, w_E)$. Two effective ways of solving the former problem is either discretizing the state space or directly sampling trajectories from the continuous space. Previously both approaches [2, 5, 6, 8, 33] have been studied. However, for high DoF manipulators sampling based approaches [5, 8] maintains tractability of the problem, hence we take this approach. More precisely, similar to Berg et al. [5], we sample trajectories using rapidly-exploring random tree (RRT) [21].[3] Since our primary goal is to learn a score function on sampled set of trajectories we now describe our learning algorithm and for more details on sampling trajectories we refer interested readers to [11, 13].

[3]When RRT becomes too slow, we switch to more efficient bidirectional-RRT. For even faster sampling one could use parallel samplers, e.g., [14]. It is important to note that sampling based methods (including ours) guarantees optimal solution modulo the sampled set and not the globally optimal solution, which is generally infeasible to find. The cost function (or its approximation) learned can be fed to trajectory optimizers like CHOMP [26] or optimal planners like RRT* [17] to produce reasonably good trajectories.

4.4. Learning the Scoring Function

Algorithm 1 Trajectory Preference Perceptron. (TPP)

Initialize $w_O^{(1)} \leftarrow 0$, $w_E^{(1)} \leftarrow 0$
for $t = 1$ to T **do**
  Sample trajectories $\{y^{(1)}, ..., y^{(n)}\}$
  $y_t = argmax_y s(x_t, y; w_O^{(t)}, w_E^{(t)})$
  Obtain user feedback $\bar{y}_t$
  $w_O^{(t+1)} \leftarrow w_O^{(t)} + \phi_O(x_t, \bar{y}_t) - \phi_O(x_t, y_t)$
  $w_E^{(t+1)} \leftarrow w_E^{(t)} + \phi_E(x_t, \bar{y}_t) - \phi_E(x_t, y_t)$
end for

The goal is to learn the parameters w_O and w_E of the scoring function $s(x, y; w_O, w_E)$ so that it can be used to rank trajectories according to the user's preferences. To do so, we adapt the Preference Perceptron algorithm [30] as detailed in Algorithm 1. We call this algorithm the Trajectory Preference Perceptron (TPP). Given a context x_t, the top-ranked trajectory y_t under the current parameters w_O and w_E, and the user's feedback trajectory $\bar{y}_t$, the TPP updates the weights in the direction $\phi_O(x_t, \bar{y}_t) - \phi_O(x_t, y_t)$ and $\phi_E(x_t, \bar{y}_t) - \phi_E(x_t, y_t)$ respectively.

Despite its simplicity and even though the algorithm typically does not receive the optimal trajectory $y_t^* = \arg\max_y s^*(x_t, y)$ as feedback, the TPP enjoys guarantees on the regret [30]. We merely need to characterize by how much the feedback improves on the presented ranking using the following definition of expected α-informative feedback: $E_t[s^*(x_t, \bar{y}_t)] \geq s^*(x_t, y_t) + \alpha(s^*(x_t, y_t^*) - s^*(x_t, y_t)) - \xi_t$. This definition states that the user feedback should have a score of $\bar{y}_t$ that is – in expectation over the users choices – higher than that of y_t by a fraction $\alpha \in (0, 1]$ of the maximum possible range $s^*(x_t, \bar{y}_t) - s^*(x_t, y_t)$. If this condition is not fulfilled due to bias in the feedback, the slack variable ξ_t captures the amount of violation. In this way any feedback can be described by an appropriate combination of α and ξ_t. Using these two parameters, the proof by [30] can be adapted to show that the expected average regret of the TPP is upper bounded by $E[REG_T] \leq \mathcal{O}(\frac{1}{\alpha\sqrt{T}} + \frac{1}{\alpha T}\sum_{t=1}^{T}\xi_t)$ after T rounds of feedback.

5. Experiments and Results

Task and Activity Set for Evaluation. We evaluate our approach on 35 different robotic tasks. For each task, we train and test on scenarios with different objects being manipulated, and/or with a different environment. We considered following three types of activities:

1) Manipulation centric: These activities primarily care for the object being manipulated. Hence the object's properties and the way robot moves it in the environment is more relevant. We consider three activities that are frequently encountered by a household robot: (a) *pick and place*, e.g., moving a glass of water without spilling water, (b) *inserting*, e.g., placing pen inside a pen holder or knife inside a knife holder, and (c) *pouring*, e.g., moving a bottle and pouring water from it into a cup. By changing the objects and environments, we create 12, 4 and 4 tasks respectively for each activity.

2) Environment centric: These activities also care for the interactions of the object being manipulated with the surrounding objects. Our object-object interaction features allow the learning algorithm to handle such type of activities. We consider two activities: (a) *heavy-fragile*, e.g., moving heavy iron box with glass table in vicinity, and (b) *liquid-electronic*, e.g., moving a glass of water with a laptop in vicinity. We create 3 tasks for heavy-fragile and 5 for liquid-electronic activity .

3) Human centric: These activities additionally have a human in the robot's vicinity. We consider activities where a robot manipulates: (a) *sharp-objects*, e.g., moving a knife with a human in vicinity, and (b) *hot-objects*, e.g., serving a hot drink. We create 2 and 5 tasks respectively for each activity.

Baseline algorithms. We evaluate the algorithms that learn preferences from online feedback, under two settings: (a) *untrained*, where the algorithms learn preferences for the new task from scratch without observing any previous feedback; (b) *pre-trained*, where the algorithms are pre-trained on other similar tasks, and then adapt to the new task. We compare the following algorithms:

- *Geometric*: It plans a single path, independent of the task, using a BiRRT [21] planner.
- *Manual*: It plans a single path following certain manually coded task dependent preferences.
- *TPP*: This is our algorithm. We evaluate it under both, *untrained* and *pre-trained* settings.
- *MMP-online*: This is an online implementation of Maximum margin planning (MMP) [28, 29] algorithm. MMP takes the maximum-margin approach to make an expert's policy better than any other policy, by a margin. It can be thought of as a special case of our algorithm with 1-informative feedback. Adapting MMP to our experiments pose two challenges (i) we do not have knowledge of optimal trajectory or demonstrations; and (ii) the state space of the manipulator we consider is too large, and discretizing will make MMP intractable. We therefore train MMP from online user feedback observed on a set of trajectories. Furthermore, the observed feedbacks are treated as optimal. At every iteration we train a structural support vector machine (SVM) [16] using all previous feedback as training examples. The learned weights are used to predict the trajectory scores for the next iteration. Since we learn on a set of trajectories, the argmax operation in structural SVM remains tractable. To quantify closeness of trajectories, we take l_2-norm of the differences in their feature rep-

resentations. We choose the regularization parameter C for training structural SVM in hindsight, to give an unfair advantage to MMP-online.

- *Oracle-svm*: This algorithm leverages the expert's labels on trajectories (hence the name *Oracle*) and is trained using SVM-rank [15] in a batch manner. This algorithm is *not realizable in practice*, as it requires labeling on the large space of trajectories. We use this only in pre-trained setting and during prediction it just predicts once and does not learn further.

Evaluation metrics. In addition to performing a user study (Section 5.2), we also designed a dataset to quantitatively evaluate the performance of our online algorithm. An expert labeled 2100 trajectories on a Likert scale of 1-5 (where 5 is the best) on the basis of subjective human preferences. (The features we designed in Section 4 were **not** revealed to the expert.) Note that these absolute ratings are never provided to our algorithms and are only used for the quantitative evaluation of different algorithms. We quantify the quality of a ranked list of trajectories by its normalized discounted cumulative gain (nDCG) [23] at positions 1 and 3 and denote them as nDCG@1 and nDCG@3 respectively. While nDCG@1 is a suitable metric for autonomous robots that execute the top ranked trajectory, nDCG@3 is suitable for scenarios where the robot is supervised by humans, (e.g., assembly lines). We also report average nDCG value over a given number of feedback iterations.

In quantitative analysis, only re-ranking feedback is used. Trajectories are observed in order of their current predicted scores and the first trajectory that is better than the top ranked trajectory is provided as feedback.

5.1. Results

How well does TPP generalize to new tasks? To study generalization of preference feedbacks we evaluate performance of TPP-pre-trained (i.e., *TPP* algorithm under *pre-trained* setting) on a set of tasks the algorithm has not seen before. We study generalization when: (a) only the object being manipulated changes, e.g., a bowl replaced by a cup or a bottle, (b) only the surrounding environment changes, e.g., rearranging objects in the environment or changing the start location of tasks, and (c) and both change. Figure 4 shows nDCG@3 plots averaged over tasks for all types of activities. TPP-pre-trained starts-off with higher nDCG@3 values than TPP-untrained in all three cases. Further, as more feedback are provided, performance of both algorithms improve and they eventually give identical performance. We further observe, an expected generalization behavior; generalizing to tasks with both new environment and object is harder than when only one of them changes. Moreover, generalizing to new environment is harder than adjusting to changes in object.

How does TPP compare to MMP-online? MMP-online proceeds by assuming every user feedback as optimal, and hence over the time accumulates many contradictory/incorrect training examples. We can see in Figure 4, pre-training MMP-online does not provide much benefit and its pre-trained and untrained versions starts-off with similar nDCG@3 values and thereon gives similar performance. This study also highlights the sensitivity of MMP to sub-optimal feedback.

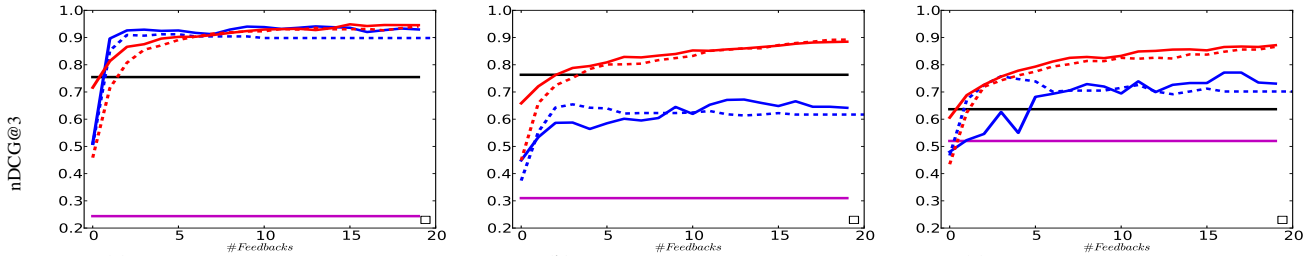
How does TPP compare to Oracle-svm? Oracle-svm starts off with nDCG@3 values higher than any other algorithm. The reason being, it is pre-trained using expert's labels on trajectories, and for the same reason it could not be used in practice. One could also observe (Figure 4), in less than 5 feedback on new task TPP improves over Oracle-svm, which is not updated since it requires expert's labels on test set.

How does TPP compare to Manual? We code some preferences into the planners e.g., keep a glass of water upright. However, some preferences are difficult to specify, e.g., not to move heavy objects over fragile items. We empirically found (Figure 4) the resultant manual algorithm produce poor trajectories with an average nDCG@3 of 0.48 over all types of activities.

How helpful are different features? Table 1 shows the performance of the TPP algorithm in the untrained setting using different features. Individually each feature capture several aspects indicating goodness of trajectories, and combined together they give the best performance. Object trajectory feature capture preferences related to the orientation of the object, which often plays a crucial role is determining preferences, e.g., moving a glass of water. Robot arm configuration and object environment feature partly capture preferences, by detecting undesirable contorted arm configurations and maintaining safe distance from surrounding surfaces, respectively. Object-object feature by itself can only learn, for example, to move water away from a laptop, but might still move it with jerks or contorted arms. This feature can be combined with other features to yield more expressive features. Nevertheless, by itself it performs better than Manual algorithm.

Table 1 also compares TPP and MMP-online under untrained setting. Since MMP-online accumulates, over time, a lot of contradictory training examples, its performance is much below TPP. On average, geometric and manual algorithms perform worse than TPP and MMP-online.

How well do variants of re-ranking feedback perform? We study the following variants of re-ranking feedback:
1. Click-one-to-replace-top: User observes the trajectories in order of their current predicted scores and clicks on the first trajectory which is better than the top ranked trajectory. We used this feedback in our quantitative analysis.
2. Click-one-from-5: Top 5 trajectories are shown and user clicks on the one he thinks is the best.
3. Approximate-argmax: In this feedback, instead of presenting top ranked trajectories, five random trajectories are



(a) Same environment, different object. (b) New Environment, same object. (c) New Environment, different object.

Figure 4: Study of generalization with change in object, environment and both. Manual, Oracle-SVM, Pre-trained MMP-online (—), Untrained MMP-online (– –), Pre-trained TPP (—), Untrained TPP (– –).

	Algorithms	Manipulation centric	Environment centric	Human centric	Mean
	Geometric	0.36 (0.54)	0.43 (0.38)	0.36 (0.27)	0.38 (0.40)
	Manual	0.53 (0.55)	0.39 (0.53)	0.40 (0.37)	0.44 (0.48)
TPP Features	Obj-obj interaction	-	0.57 (0.55)	0.53 (0.48)	0.55 (0.53)
	Robot arm config	0.89 (0.87)	0.75 (0.66)	0.64 (0.53)	0.76 (0.69)
	Object trajectory	0.92 (0.91)	0.80 (0.70)	0.71 (0.60)	0.81 (0.74)
	Object environment	0.92 (0.90)	0.77 (0.68)	0.60 (0.49)	0.76 (0.69)
	TPP (all features)	**0.93 (0.92)**	**0.85 (0.75)**	**0.78 (0.66)**	**0.85 (0.78)**
	MMP-online	0.83 (0.82)	0.42 (0.51)	0.36 (0.33)	0.54 (0.55)

Table 1: **Comparison of different algorithms and study of features in untrained setting.** Table contains average nDCG@1(nDCG@3) values over 20 rounds of feedbacks.

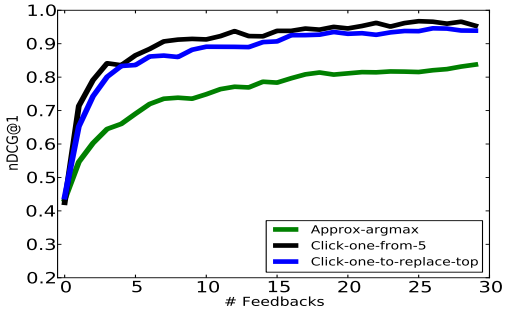


Figure 5: **Study of re-ranking feedbacks.**

selected as candidate. The user selects the best trajectory among these 5 candidates. This simulate situations when computing an argmax over trajectories is prohibitive and therefore an approximate argmax is performed.

Figure 5 shows the performance of TPP-untrained receiving different kinds of feedbacks and averaged over three types of activities. When feedback are more α-informative the algorithm requires fewer of those to learn preferences. In particular, click-one-to-replace-top and click-one-from-5 feedbacks are the most informative, while approximate-argmax is the least informative and therefore requires more feedbacks to reach a given nDCG@1 value. Since all the feedback is α-informative, for some $\alpha > 0$, eventually TPP-untrained is able to learn the preferences. These empirical findings are consistent with the regret bounds presented in section 4.4.

5.2. User Study in learning trajectories

We perform a user study of our system on a variety tasks of varying difficulty, showing it is practically realizable, and in untrained setting it learns in few feedbacks.

Experiment setup: In this study, two users used our system to train a robot, using re-ranking and interactive feedback, in accordance with their preferences. A set of 10 tasks of varying difficulty level was presented to them one at a time, and users were instructed to provide feedback until they were not satisfied with the top ranked trajectory. As a baseline Oracle-svm was trained on related tasks and evaluated on those 10 tasks. To quantify the quality of learning, each user also evaluated their own trajectories (self score), the trajectories of the other user (cross score), and those predicted by Oracle-svm, on a Likert scale of 1-5. Users were not revealed the trainer of the trajectories they were assessing.

Analysis: The correlation in user's self and cross scores, Table 2, suggest both users shared similar utility functions. Furthermore, on an average a user took 3 re-ranking and 2 interactive feedback to train an untrained robot. We conjecture, for a high DoF manipulator, re-ranking feedback are easier to provide than interactive – which requires modifying the manipulator joint angles.

Learning curve: On average, TPP took 5 feedback to improve over Oracle-svm, Figure 6 (Left). This is consistent with our quantitative analysis in the previous section.

Task difficulty: A trend of increase in number of interactive feedback with increase in total number of feedback was observed, Figure 6 (Right). We conjecture from this, interactive feedback were more informative than re-ranking and user relied more on them for difficult tasks.

6. Conclusion

In this paper we presented a co-active learning framework for training robots to select trajectories that obey a user's preferences. Unlike in standard learning from demonstration approaches, our framework does not require the user to provide optimal trajectories as training data, but can learn from iterative improvements. Despite the weak feedback requirements, our TPP learning algorithm has provable re-

Table 2: **Learning statistics for each user.** Self and cross scores of the final learned trajectories. The number inside bracket is standard deviation.

User	# Re-ranking feedbacks	# Interactive feedbacks	Trajectory Quality self	Trajectory Quality cross
1	3.1 (1.3)	2.4 (2.4)	3.5 (1.1)	3.6 (0.8)
2	2.3 (1.1)	1.8 (2.7)	4.1 (0.7)	4.1 (0.5)

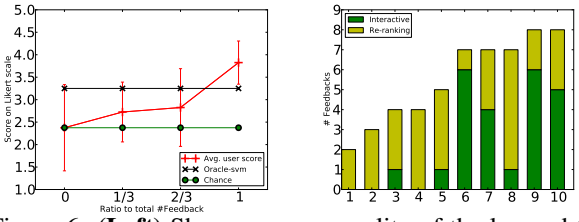


Figure 6: **(Left)** Shows average quality of the learned trajectory after every one-third of total feedbacks. **(Right)** Bar chart showing the average number of feedbacks (re-ranking and interactive) required for each task. Task difficulty increases from 1 to 10.

gret bounds and empirically performs well. In particular, we propose a set of trajectory features for which the TPP generalizes well on tasks which the robot has not seen before, as well as in a transfer learning setting where either the object, the environment, or both were changed. In addition to the batch experiments, a user study confirmed that incremental feedback generation is indeed feasible and that it leads to good learning results already after only a few iterations.

References

[1] Abbeel, P., Coates, A., and Ng, A. Y. Autonomous helicopter aerobatics through apprenticeship learning. *IJRR*, 29 (13), 2010.

[2] Alterovitz, R., Siméon, T., and Goldberg, K. The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. In *RSS*, 2007.

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

[4] Berenson, D., Abbeel, P., and Goldberg, K. A robot path planning framework that learns from experience. In *ICRA*, 2012.

[5] Berg, J. V. D., Abbeel, P., and Goldberg, K. Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information. In *RSS*, 2010.

[6] Bhattacharya, S., Likhachev, M., and Kumar, V. Identification and representation of homotopy classes of trajectories for search-based path planning in 3d. In *RSS*, 2011.

[7] Cohen, B. J., Chitta, S., and Likhachev, M. Search-based planning for manipulation with motion primitives. In *ICRA*, pp. 2902–2908, 2010.

[8] Dey, D., Liu, T. Y., Hebert, M., and Bagnell, J. A. Contextual sequence prediction with application to control library optimization. In *RSS*, 2012.

[9] Diankov, R. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, CMU, RI, August 2010.

[10] Dragan, A., Lee, K., and Srinivasa, S. Legibility and predictability of robot motion. In *HRI*, 2013.

[11] Erickson, L. H. and LaValle, S. M. Survivability: Measuring and ensuring path diversity. In *ICRA*, 2009.

[12] Gossow, D., Hershberger, A. Leeperand D., and Ciocarlie, M. Interactive markers: 3-d user interfaces for ros applications [ros topics]. *Robotics & Automation Magazine, IEEE*, 18(4):14–15, 2011.

[13] Green, C. J. and Kelly, A. Toward optimal sampling in the space of paths. In *Robotics Research*. 2011.

[14] Ichnowski, J. and Alterovitz, R. Parallel sampling-based motion planning with superlinear speedup. In *IROS*, 2012.

[15] Joachims, T. Training linear svms in linear time. In *KDD*, 2006.

[16] Joachims, T., Finley, T., and Yu, C.-N. J. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, 2009.

[17] Karaman, S. and Frazzoli, E. Incremental sampling-based algorithms for optimal motion planning. In *RSS*, 2010.

[18] Kober, J. and Peters, J. Policy search for motor primitives in robotics. *Machine Learning*, 84(1):171–203, 2011.

[19] Koppula, H. and Saxena, A. Anticipating human activities using object affordances for reactive robotic response. In *RSS*, 2013.

[20] Koppula, H.S., Anand, A., Joachims, T., and Saxena, A. Semantic labeling of 3d point clouds for indoor scenes. In *NIPS*, 2011.

[21] LaValle, S. M. and Kuffner, J. J. Randomized kinodynamic planning. *IJRR*, 20(5):378–400, 2001.

[22] Levine, S. and Koltun, V. Continuous inverse optimal control with locally optimal examples. In *ICML*, 2012.

[23] Manning, C. D., Raghavan, P., and Schütze, H. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, 2008.

[24] Phillips, M., Cohen, B., Chitta, S., and Likhachev, M. E-graphs: Bootstrapping planning with experience graphs. In *RSS*, 2012.

[25] Ratliff, N., Bradley, D., Bagnell, J. A., and Chestnutt, J. Boosting structured prediction for imitation learning. In *NIPS*, 2007.

[26] Ratliff, N., Zucker, M., Bagnell, J. A., and Srinivasa, S. Chomp: Gradient optimization techniques for efficient motion planning. In *ICRA*, 2009.

[27] Ratliff, N. D. *Learning to Search: Structured Prediction Techniques for Imitation Learning*. PhD thesis, CMU, RI, 2009.

[28] Ratliff, N. D., Bagnell, J. A., and Zinkevich, M. Maximum margin planning. In *ICML*, 2006.

[29] Ratliff, N. D., Silver, D., and Bagnell, J. A. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53, 2009.

[30] Shivaswamy, P. and Joachims, T. Online structured prediction via coactive learning. In *ICML*, 2012.

[31] Sucan, I. A., Moll, M., and Kavraki, L. E. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012. `http://ompl.kavrakilab.org`.

[32] Tamane, K., Revfi, M., and Asfour, T. Synthesizing object receiving motions of humanoid robots with human motion database. In *ICRA*, 2013.

[33] Vernaza, P. and Bagnell, J. A. Efficient high dimensional maximum entropy modeling via symmetric partition functions. In *NIPS*, 2012.

[34] Wilson, A., Fern, A., and Tadepalli, P. A bayesian approach for policy learning from trajectory preference queries. In *NIPS*, 2012.

[35] Zacharias, F., Schlette, C., Schmidt, F., Borst, C., Rossmann, J., and Hirzinger, G. Making planned paths look more human-like in humanoid robot manipulation planning. In *ICRA*, 2011.

[36] Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. Maximum entropy inverse reinforcement learning. In *AAAI*, 2008.