Graph-Based Inverse Optimal Control for Robot Manipulation

Arunkumar Byravan[1], Mathew Monfort[2], Brian Ziebart[2], Byron Boots[3] and Dieter Fox[1*]

Abstract

Inverse optimal control (IOC) is a powerful approach for learning robotic controllers from demonstration that estimates a cost function which rationalizes demonstrated control trajectories. Unfortunately, its applicability is difficult in settings where optimal control can only be solved approximately. While local IOC approaches have been shown to successfully learn cost functions in such settings, they rely on the availability of good reference trajectories, which might not be available at test time. We address the problem of using IOC in these computationally challenging control tasks by using a graph-based discretization of the trajectory space. Our approach projects continuous demonstrations onto this discrete graph, where a cost function can be tractably learned via IOC. Discrete control trajectories from the graph are then projected back to the original space and locally optimized using the learned cost function. We demonstrate the effectiveness of the approach with experiments conducted on two 7-degree of freedom robotic arms.

1 Introduction

Robotic manipulation in environments shared with humans is difficult. Robots must not only succeed with their well-specified task objectives, such as grasping and placing objects, but they must also produce motion trajectories that satisfy criteria that are difficult to specify. For example, one might want a robot to hold a cup as upright as possible while avoiding collisions with other objects or to avoid moving the cup above an expensive electronic item such as a laptop (see Fig. 1). Despite being able to easily recognize and demonstrate such trajectories, specifying evaluation criteria that produce them might be difficult for the roboticist. Recent advances in planning [Zucker *et al.*, 2013; Schulman *et al.*, 2013] have proven to be very successful at fulfilling objectives such as being collision-free or generating smooth motion, but often these trajectories tend to ignore human preferences.

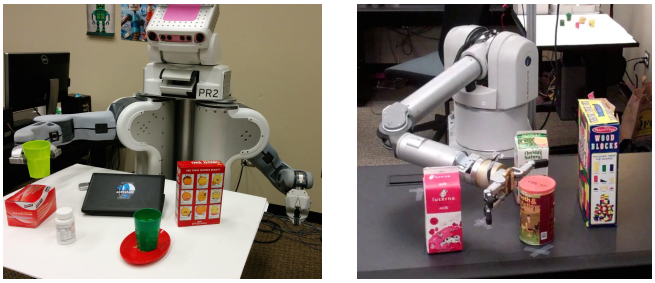


Figure 1: PR2 robot and Barrett WAM arm used in our experiments. The manipulators have 7 degrees of freedom, posing a challenge for learning cost functions for trajectory planners.

Inverse optimal control (IOC) [Abbeel and Ng, 2004; Ziebart *et al.*, 2008; Ng and Russell, 2000; Ratliff *et al.*, 2009; Levine and Koltun, 2012] provides an attractive option for learning motion trajectory criteria that are difficult for the roboticist to specify. IOC estimates a cost function for a decision process that (partially) rationalizes manipulation trajectories demonstrated via teleoperation by making them (near) optimal. The key advantage of IOC—as opposed to directly estimating a control policy—is that this estimated cost function generalizes to new situations. For example, if obstacles are re-arranged, a low-cost trajectory will avoid collisions while replaying a previous trajectory will not. IOC has been employed to construct controllers and planners for simulated humanoid control [Levine and Koltun, 2012], field robot navigation [Ratliff *et al.*, 2009], navigation through crowds [Henry *et al.*, 2010], and goal prediction for robotic manipulation [Dragan *et al.*, 2011].

The many degrees of freedom (DOF) of typical robotic arms pose significant challenges for using inverse optimal control on manipulation tasks. IOC relies on solving the optimal control problem, but this is generally intractable and state-of-the-art planning methods provide no optimality guarantees. Furthermore, to robustly learn cost functions from non-optimal demonstrations requires the ability to reason about distributions over paths, rather than only computing the optimal one [Ziebart *et al.*, 2008]. For the specific case of sys-

*[1]Department of Computer Science & Engineering, University of Washington, [2]Department of Computer Science, University of Illinois at Chicago, [3]School of Interactive Computing, Georgia Institute of Technology

tems with linear dynamics and quadratic cost functions, efficient inverse optimal control methods have been developed [Levine and Koltun, 2012; Ziebart *et al.*, 2012]. However, these assumptions rarely hold in manipulation tasks. The desirable space of manipulation trajectories is often multimodal and non-linear due to collision avoidance with discrete obstacles. Despite this, IOC methods have leveraged the special case of linear-quadratic systems by locally rationalizing demonstrated trajectories using a linear-quadratic approximation centered around the demonstrated trajectory itself [Levine and Koltun, 2012]. Unfortunately, when a controller applies the resulting cost function to a new situation, the correct reference trajectory for linear-quadratic approximation is unknown. Other reference trajectories may produce inherently non-human-acceptable manipulation trajectories.

Motivated by the weaknesses of existing IOC methods for handling higher-dimensional control tasks, we propose an IOC approach that is more complementary to the strengths of recent robotic trajectory planners. Our key observation is that trajectories produced by planners often fail to be human-acceptable due to their coarse characteristics (e.g., the direction of approach or choice of paths weaving between obstacles) rather than their fine-scale characteristics (e.g., smoothness, obstacle avoidance). Our approach uses IOC to learn preferences for trajectories through a coarse, discrete, graph-based approximation of the trajectory space and then uses a trajectory optimizer to fine-tune the resulting coarse trajectory based on the learned cost function. We demonstrate the effectiveness of this approach with several experiments conducted on a 7-DOF robotic arm.

2 Background and Related Work

2.1 Trajectory planning

Motion planning based on trajectory optimization has recently been very successful in solving hard, high dimensional planning problems. Algorithms like CHOMP [Zucker *et al.*, 2013] and STOMP [Kalakrishnan *et al.*, 2011] generate collision-free paths by minimizing a cost function based on trajectory smoothness and distance, while TrajOpt [Schulman *et al.*, 2013] uses sequential convex optimization with collision avoidance constraints.

Due to the inherent computational difficulties of optimization over the space of trajectories, these methods have some drawbacks. They compute locally optimal trajectories with no bounds on the global sub-optimality of the resulting solution. Further, they often produce trajectories that, while smooth and able to accomplish the specified objective, differ substantially from human-produced trajectories.

2.2 Maximum entropy inverse optimal control

Maximum entropy inverse optimal control (MaxEnt IOC) [Ziebart *et al.*, 2008] combines reward-based guarantees of inverse reinforcement learning [Abbeel and Ng, 2004] with predictive guarantees of robust estimation [Topsøe, 1979; Grünwald and Dawid, 2004]. It obtains the stochastic policy that is least biased while still matching feature counts [Abbeel and Ng, 2004]. Its resulting predictive policy estimate, $P(a_t|s_t) \propto e^{Q(s_t,a_t)}$, is recursively defined using a softened version of the Bellman equation:

$$Q(s_t, a_t) \triangleq \mathbb{E}_{P(s_{t+1}|a_t,s_t)}[\operatorname*{softmax}_{a_{t+1}} Q(S_{t+1}, a_{t+1})] - \text{cost}_{\theta,\mathbf{f}}(s_t, a_t) \quad (1)$$

where $\text{cost}_{\theta,\mathbf{f}}(s_t, a_t) \triangleq \theta^{\mathrm{T}}\mathbf{f}(a_t, s_t)$, $\operatorname{softmax}_x f(x) = \log \sum_x e^{f(x)}$ and $\mathbf{f}(a_t, s_t)$ is a vector of features characterizing the state-action pair. The term $\theta^{\mathrm{T}}\mathbf{f}(a_t, s_t)$ is analogous to the cost function of optimal control. Parameters θ are estimated by maximizing the training data likelihood under the MaxEnt distribution, which for deterministic planning settings are Boltzmann distributions over state sequences:

$$P(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}) = \frac{e^{-\theta^{\mathrm{T}} \sum_{t=1}^{T} \mathbf{f}(s_t,a_t)}}{\sum_{\mathbf{s}'_{1:T},\mathbf{a}'_{1:T}} e^{-\theta^{\mathrm{T}} \sum_{t=1}^{T} \mathbf{f}(s'_t,a'_t)}} \propto e^{-\text{cost}_\theta(\mathbf{s}_{1:T},\mathbf{a}_{1:T})}. \quad (2)$$

MaxEnt IOC has been developed for both discrete and continuous state and action settings. We detail both settings here.

Discrete: Discrete state-action representations can incorporate arbitrary dynamics and features, but are limited by the $\mathcal{O}(|S||A|T)$ complexity of dynamic programming algorithms that compute Eqn.1. This limits the approach to low-dimensional settings.

Continuous: Eqn. 1 can be analytically solved for continuous state-action representations with linear dynamics, $\vec{s}_{t+1} = \mathbf{A}\vec{s}_t + \mathbf{B}\vec{a}_t$, and quadratic features, $\text{cost}(\vec{s}_{1:T}) = \sum_{t=1}^{T} \vec{s}_t^{\mathrm{T}}\mathbf{\Theta}\vec{s}_t$, with parameter matrix $\mathbf{\Theta}$, [Ziebart, 2010; Ziebart *et al.*, 2012; Levine and Koltun, 2012].

2.3 Learning from robotic motion trajectories

Ideally one would learn cost functions for state-of-the-art trajectory planning algorithms to generate human-like trajectories that generalize sufficiently well across tasks. Unfortunately, the non-convexity and discontinuity of the trajectory planner's solutions' costs as a function of cost parameters poses significant theoretical challenges for IOC. Standard gradient-based optimization methods cannot be reliably employed to optimize the cost parameters of such functions since local optima created by these discontinuities lead to inappropriate parameter estimates in both theory and practice.

One approach is to locally approximate the dynamics and cost function as a linear-quadratic model around a reference trajectory [Levine and Koltun, 2012] and use continuous MaxEnt IOC. Path integral methods for imitation learning [Aghasadeghi and Bretl, 2011; Kalakrishnan *et al.*, 2013] take a similar form, but impose a cost on controls that is inverse to the amount of control noise. In each case, locally optimizing by approximating around the demonstrated trajectory makes learning possible. However, finding an appropriate reference trajectory when needing to produce a trajectory for a new situation is difficult and remains an open problem. Though co-active learning methods for trajectory imitation [Jain *et al.*, 2013] learn from trajectory refinements rather than full trajectory demonstrations, they suffer from similar local optimization limitations. We avoid these concerns by learning at a coarse granularity for which optimal control and reasoning about distributions of discrete paths remains tractable.

3 Approach

To create a trajectory planner capable of producing trajectories that are more acceptable to people, we begin by separating the manipulation trajectory planning task into two distinct problems: (1) coarsely choosing a natural motion for the trajectory based on relational and topological properties with objects and obstacles; and (2) refining the coarse trajectory to be smooth and precise near obstacles. This partitions the desirable properties of manipulation trajectories that are difficult to specify—the topological "naturalness" of the trajectory—from the desirable obstacle avoidance and smoothness properties that are well-specified and solved by recent trajectory planning methods.

We coarsely approximate the space of manipulation trajectories using a discrete graph representation and employ MaxEnt IOC [Ziebart *et al.*, 2008] to learn complex cost functions that partially rationalize demonstrated manipulation trajectories that we project onto this graph. A generalizable parametric probability distribution over paths through the graph is also produced by the MaxEnt IOC approach. We employ local trajectory optimization on samples from this graph distribution to produce smooth, obstacle avoiding trajectories using criteria that do not need to be learned. The simplifications of our discrete representation intentionally ignore important aspects of the manipulation task that are computationally difficult to fully incorporate; for instance, it does not employ strict collision detection with certain obstacles. Key then is finding a representation of the manipulation task that balances computational tractability against the realizable similarity between demonstrated trajectories and the estimated trajectory distribution. We now present details on the two major components of our approach: the discrete IOC model and the local trajectory optimization of sampled discrete paths.

3.1 Discrete IOC in a coarse path space

We construct a sparse discrete space to tractably approximate our continuous trajectory space. This discrete space is represented as a graph in the robot's configuration space ($\mathcal{Q}$). We project the demonstrations onto this graph and use discrete MaxEnt IOC to learn a distribution over graph paths that matches the projections.

A key consideration in our approach is the ability to generalize to new situations. For this reason, shaping the construction of the graph based on the demonstrations must be avoided as those trajectories will not be available when generating trajectories for new situations. More specifically, to avoid the problems that arise from locally-optimized inverse optimal control methods [Levine and Koltun, 2012], we construct our graph representation of the trajectory space independently from the specific trajectories used for training.

Graph generation: Algorithm 1 explains the procedure for generating the sparse discrete graph. Initially, we compute a diagonal co-variance Σ based on the range of motion (ξ_e) of the robot's DOF, restricting the maximum range to 2π. The DOF with the maximum range is assigned a unit variance. To avoid sampling uniformly over the robot's configuration space, we make an assumption that the demonstrations tend to lie close to the straight line trajectory. Given a start and a goal ($\xi_s, \xi_g \in \mathcal{Q}$) configuration, we discretize the straight line trajectory ($\xi_s \rightarrow \xi_g$) into m waypoints. We generate samples from Gaussians centered around these m waypoints.

Algorithm 1 Generating a graph $\mathcal{G}$

Inputs: ξ_s, ξ_g, m, M, k, ξ_{high}, ξ_{low}, σ
Output: Graph, $\mathcal{G} = (V, E)$
$V = \{\}, E = \{\}$
ξ_e = min($|\xi_{high} - \xi_{low}|, 2\pi$) ▷ Compute joint extents
$\Sigma = diag(\frac{\xi_e}{\max(\xi_e)})$ ▷ Diagonal co-variance
for $i = 0 : m-1$ **do**
  $\xi_i = \xi_s + \frac{i}{m-1}(\xi_g - \xi_s)$ ▷ Linear interpolation
  $n_i \approx M * \mathcal{N}(i|\frac{m-1}{2}, \sigma)$ ▷ $\sum_{j=0}^{m-1} n_i = M$
  $\Sigma_i \approx \Sigma * \mathcal{N}(i|\frac{m-1}{2}, \sigma)$ ▷ Scale co-variance
  for $j = 1 : n_i$ **do**
    $\xi_j \in \mathcal{N}(\xi_i, \Sigma_i)$ ▷ Sample robot configuration
    $V = V + \xi_j$ ▷ Add to vertex set
E = NN(V, k) ▷ k-Nearest Neighbour set

While the demonstrations are close to the straight line near the endpoints, the uncertainty increases as we traverse along the trajectory. Our graph construction handles this in two ways: First, we increase the co-variance of the Gaussians near the center and proportionally make the Gaussians near the endpoints tight (Fig. 2). Second, proportional to the increase in co-variance, we also sample more points from the Gaussians near the center. The scaling factor is defined by a one-dimensional Gaussian kernel centered around the midpoint $\frac{m-1}{2}$, with a variance σ. In total, we sample M points from the Gaussians. Each node is then connected to its k-Nearest Neighbors (NN) to form an undirected graph $\mathcal{G}$ (Fig. 3).

Projecting demonstrations to the graph: We project trajectories into the space of paths through our coarse graph ($\mathcal{G}$) by computing the path through the graph that most closely matches the demonstrated trajectory where we define closeness via Euclidean distance in $\mathcal{Q}$. We accomplish this using a modified version of Dijkstra's algorithm that finds the shortest path on the graph that is close to the demonstrated trajectory. At each parent vertex in the graph V_p, the algorithm tries to create a path through a neighbor V_j that is close to the demonstration and minimizes the transition cost:

$$V_n = \underset{V_j \in E(V_p)}{\arg\min}\ \gamma||V_j - V_p||_2 + \min_{k=[p,\ldots,g]}(||V_j - \xi_k||_2) \quad (3)$$

The first term in Eqn. 3 is the transition cost between vertices - Euclidean distance between configurations. The second term is the distance between the candidate vertex V_j and its closest point on the *remaining* part of the demo $[\xi_p, ..., \xi_g]$, where ξ_p is the closest point on the trajectory to the parent vertex (V_p). At the start vertex V_s, this is ξ_s. The algorithm continues until it finds a path to the goal vertex V_g. Fig.3 shows the demonstration and its projection for a 2D example.

Discrete MaxEnt IOC: Given our coarse graph, we estimate a distribution over paths through the graph that has maximum entropy while matching certain properties of the projected demonstrations. These properties are characterized by task-dependent features, $\sum_t f(\xi_t)$, which, for computational reasons, are based solely on the states of the graph (i.e., joint configurations). Equivalently, this distribution results from

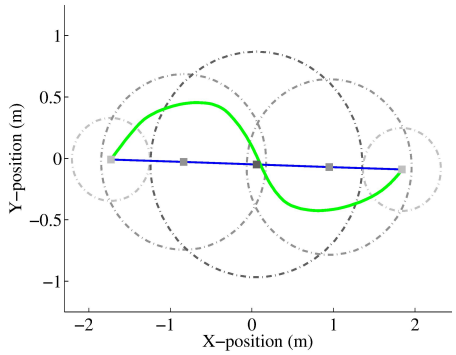


Figure 2: Gaussian ellipses around waypoints on a straight line path (Blue) between two points in 2D. Demonstrated trajectory shown in green. Larger ellipses (darker) near center have larger covariances.

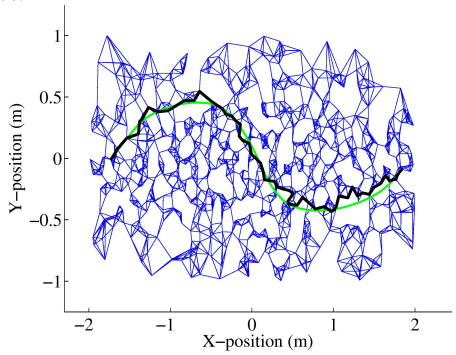


Figure 3: Sampled graph with 1000 nodes and 5NN edges (blue) with the projection (black) of demonstrated trajectory (green).

maximizing the likelihood of the demonstrated trajectory projections under the MaxEnt distribution (Eqn. 2):

$$\theta^{\star} = \arg\max_{\theta} \mathcal{L}(\xi^{demo}_{1:T}) = \arg\max_{\theta} \frac{e^{-\theta^{\mathrm{T}} \sum_{t=1}^{T} \mathbf{f}(\xi_t^{demo})}}{\sum_{\xi'_{1:T}} e^{-\theta^{\mathrm{T}} \sum_{t=1}^{T} \mathbf{f}(\xi'_t)}}. \tag{4}$$

We estimate parameters as linear weights (θ) for the feature vectors ($\mathbf{f}$). The resulting potentials ($\theta^T \mathbf{f}$) can be interpreted as analogs to state "costs." As (2) is concave in terms of θ, the optimal weights ($\theta^{\star}$) are found using standard gradient based optimizers, making use of efficient softmax inference (Eqn. 1) via an optimized value iteration algorithm for gradient and likelihood computation.

An important property of the MaxEnt IOC approach is its unbiasedness. Beyond the learned cost function, its predictions are as agnostic as possible. Thus, if additional trajectory properties are important, but not captured by the coarse approximation of the trajectory space, samples from the MaxEnt IOC distribution will often provide coverage—though many samples may need to be taken depending on the complexity of the additional properties.

Sampling paths from discrete graph: We sample paths from our discrete graph in two ways. First, motivated by MaxEnt IOC's unbiasedness to preferences over unknown features, we probabilistically sample goal-directed paths directly from our learned path distribution. Alternatively, we deterministically find the most probable path, which minimizes the cost of traversal through the graph.

3.2 Local Trajectory Optimization

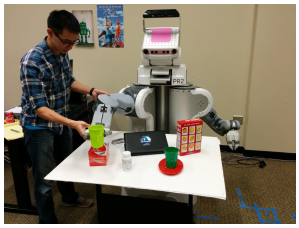
The paths sampled from the discrete graph based on the learned cost typically avoid obstacles and match the desirable properties of the human demonstrations. However, due to the sparsity of sampling in the high-dimensional space, these paths are not sufficiently smooth to be executed by the manipulator, and they might not completely match the properties of the demonstrations. For instance, when learning to hold a cup upright, the discrete graph might not contain nodes that correspond exactly to upright poses of the end effector. Fortunately, we can resort to a Local Trajectory Optimizer (LTO) to generate smooth, continuous trajectories from the discrete paths. Our approach locally optimizes around a sampled path using the learned cost function while enforcing smoothness.

The trajectory optimizer is inspired by the CHOMP [Zucker *et al.*, 2013] motion planner. We represent a trajectory $\xi \in \Xi$ as a set of n waypoints in the robot's configuration space ($\xi = [\xi_1, \xi_2, ...\xi_n];\ \xi_i \in \mathcal{Q}$) and define a cost function $\mathcal{U} : U(\xi) \in \mathcal{R}$ that assigns a cost to each trajectory ξ: $\mathcal{U}(\xi) = \eta \mathcal{U}_{smooth}(\xi) + \mathcal{U}_{learned}(\xi)$. The smoothness cost ($\mathcal{U}_{smooth}$) measures the shape of the trajectory and is defined as an integral over squared tangent norms (we use squared acceleration). The second term in the cost function ($\mathcal{U}_{learned}$) corresponds to the cost function learned via discrete IOC: $\mathcal{U}_{learned} = \sum_{i=1}^{n} \theta^{\star T} f(\xi_i)$. By minimizing the combined cost $\mathcal{U}$, LTO tries to find a smooth trajectory that has low cost under the learned cost function, i.e., one that captures the user preferences as highlighted by the demonstrations. LTO optimizes the cost via gradient descent. At each iteration of the optimization, it computes a first-order Taylor series expansion of the cost around the current trajectory (ξ^t):

$$\mathcal{U}(\xi) = \mathcal{U}(\xi^t) + \nabla \mathcal{U}(\xi^t)^T (\xi - \xi^t) + \lambda ||\xi - \xi^t||_{\mathcal{A}} \tag{5}$$

where the regularization is with respect to an admissible norm. In our work, the norm $\mathcal{A}$ measures squared accelerations, constraining consequent iterates to have similar acceleration profiles. The update rule is given by: $\xi^{t+1} = \xi^t - \frac{1}{\lambda} \mathcal{A}^{-1} \nabla \mathcal{U}(\xi^t)$. This update requires the gradient of the combined cost function $\mathcal{U}$. The gradient of the smoothness cost $\nabla \mathcal{U}_{smooth}$ can be computed analytically, as shown for CHOMP. However, while CHOMP has been developed for analytically differentiable cost functions, our learned cost functions $\mathcal{U}_{learned}$ are significantly more complex and typically non-differentiable, depending on the features. We thus have to resort to finite differencing to compute the gradient $\nabla \mathcal{U}_{learned}$ of the learned cost. To ensure that the optimization is well conditioned, we reduce the gradient descent step-size ($\frac{1}{\lambda}$) and set a high weight (η) for the smoothness cost. The final trajectory $\xi^{\star}$ at the end of the optimization is smooth, continuous, and has low cost under the learned cost function.

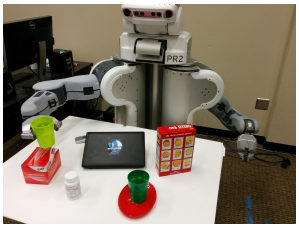
In summary, our approach generates discrete graphs to learn complex cost functions from multiple demonstrations using IOC. During testing, the approach first generates a discrete graph connecting a pair of given start and goal points in the high-dimensional joint space of the manipulator. It then



(a) Demonstration

(b) Right-Approach

(c) Left-Approach



(d) Laptop

Figure 4: Experiments with the PR2 robot. (a) Demonstrations for training and testing were generated by moving the robot's arm given a task description. (b,c) Typical pose on a trajectory when tasked to approach the target object from right or left side, respectively. (d) Example when holding the cup upright while not moving it above the laptop.

computes the feature values and resulting cost for all points on the graph, followed by running Dijkstra's algorithm to determine the lowest cost path or to sample multiple paths according to the corresponding path distribution. These paths are then used to initialize the local trajectory optimizer, which generates smooth paths while also considering the learned cost function. From the possibly multiple paths, the robot executes the one that has the lowest overall cost.

4 Evaluation

4.1 Setup

We evaluate our approach on three manipulation tasks using the 7-DOF manipulators Barrett WAM and PR2 (Fig. 1).
L/R Approach: Approach an object from either the right (Fig. 4b) or left (Fig. 4c).
Move-can: Carry a can to a goal while holding it upright.
Laptop : Carry a cup of water upright, while not moving it over electronic objects (Fig. 4d).
For each task, we collected demonstrations from multiple users via kinesthetic teaching (Fig. 4a). We record the joint angles and the various object positions (segmented point clouds and bounding boxes) for each of the demonstrations. On average, we collected 50 demonstrations for each task from four different users.
Features: We learn using the following state-dependent, binary feature functions ($\approx$100 total), $f(\xi_i)$:
Collision: Most of the features are related to distances from objects in the environment. We segment the point cloud from a Kinect mounted on the robot to generate clusters. For fast distance computation, we represent the robot as a collection of spheres, approximate the object clusters by bounding boxes and compute object-centric signed distance functions [Curless, 1997] to represent obstacles. We compute the minimum distance to any obstacle, the minimum distance from the end-effector to task-dependent target objects, and the average distances from obstacles for the end-effector and other parts of the robot. We use histograms of all these distances as our features. We also measure self-collision distances and use histograms of these as features.
Elbow clearance: We use histograms of elbow clearance from the table-top to capture elbow-up/down preferences.
End-Effector orientation: We use histograms of the deviation of the end-effector's normal vector from the vertical, and histograms of the last two joint angle differences to capture constraints based on the end-effector orientation.
Approach direction: To capture the target object approach direction, we compute histograms of the difference between the end effector's position and the center of the target object.
Laptop feature: We detect any electronic objects (laptop, remote) in the scene by matching the object clusters against pre-computed models. We add a feature indicating the end-effector is above electronic objects and one for the end-effector being above any other object in the scene.
Constant feature: Finally, we add a constant feature to match the path length of the demonstrations.
Algorithms: We evaluate the properties and advantages of our algorithm using comparisons on held out test data:
Human: Evaluation metrics applied to the human demonstrations for the held out scenes.
CHOMP+Obstacle avoidance: Path generated by running the CHOMP motion planner. This only serves as a baseline that measures the difficulty of the learning problem.
Least Cost graph path: Least cost graph path using the learned cost function on the test scene. We typically interpolate it by 10x before testing.
LTO+Random paths: Our Local Trajectory Optimizer (LTO) initialized with a straight line and smooth paths randomly generated from a gaussian around the straight line path.
LTO+Least Cost graph path: LTO initialized with the least cost discrete path from the graph.
LTO+Sampled graph paths: LTO initialized with multiple discrete graph paths sampled using the learned cost function.
Additionally, we tested a ***locally-optimal*** IOC algorithm that learns the cost function by matching the features of the locally-optimal continuous trajectory (computed directly using LTO without any graph) with that of the demonstrations, similar to an MMP-like setting [Ratliff *et al.*, 2009].

4.2 Parameters

We split the demonstrations into a training set (70%) for learning the cost function and a held out test set (30%) to measure performance. We discretize the straight line path into $m = 21$ points and generate a graph with $M = 210000$ nodes and $k = 15$. We initialize the weights (θ) randomly and iterate until convergence. We use two sets of parameters for the Local Trajectory Optimizer. For tasks with constrained motions, we set $\eta = 10$ and step-size $\frac{1}{\lambda} = 0.01$ and for larger

	Left/Right Approach		Move-Can		Laptop		
	In Coll. (%)	Wrong side (%)	In Coll. (%)	Norm. Dev (deg)	In Coll. (%)	Norm. Dev (deg)	Above Laptop (%)
Human	0.5	33 ± 2.1	1.9	9.4 ± 0.6	**2.7**	7.4 ± 0.5	2.1 ± 0.9
CHOMP + Obstacle avoidance	**0.4**	57 ± 2.9	**0.1**	13.0 ± 0.8	12.9	18.2 ± 1.7	17.3 ± 3.3
Least Cost graph path	5.1	45 ± 2.9	8.9	10.8 ± 0.6	12.8	9.9 ± 0.5	11.1 ± 2.3
LTO + Random paths	1.4	27 ± 1.9	4.7	8.9 ± 0.2	4.5	5.5 ± 0.4	3.1 ± 1.0
LTO + Least Cost graph path	2.4	29 ± 2.2	1.9	8.8 ± 0.2	6.1	5.4 ± 0.4	4.3 ± 1.5
LTO + Sampled graph paths	1.2	$\mathbf{25 \pm 1.6}$	0.5	$\mathbf{8.6 \pm 0.2}$	4.0	$\mathbf{5.3 \pm 0.4}$	$\mathbf{1.2 \pm 0.5}$

Table 1: Learning performance on the withheld test set. Error bars indicate standard error, and best results are highlighted in bold font. Metrics are: a) *In Coll*: Percentage of trajectory points in collision, b) *Wrong side*: Percentage of points where the robot's end-effector is to the wrong side of the target object, c) *Norm Dev*: Average deviation of the end-effector's normal from the vertical and d) *Above Laptop*: Percentage of points where end-effector passes above electronic objects.

motions, we set $\eta = 4$ and $\frac{1}{\lambda} = 0.025$ for aggressive optimization. We run LTO for 100 iterations. For initialization, we use the least cost discrete path and 25 discrete path samples (interpolated 10x) from the graph, based on the learned cost. For CHOMP, we use a fixed step-size (0.005), $0.1m$ collision threshold, unit smoothness and obstacle weights and 200 iterations. Both CHOMP and LTO+Random are initialized with the same random Gaussian samples.

4.3 Results

Table 1 summarizes the results on the three tasks.

L/R Approach: The first column for this task presents the overall percentage of trajectory points where the robot is in collision with objects or itself. As can be seen, all techniques achieve very low percentages. Not surprisingly, CHOMP has the lowest collision percentage since its only goal is to smoothly get to the target while avoiding obstacles. Our approach, on the other hand, is not given explicit obstacle avoidance constraints and has to learn to avoid obstacles from the demonstrations. The least cost graph path has highest collision percentage since we simply interpolate linearly between points on the graph, which often produces additional points in collision. The second column measures the percentage of trajectory points that are on the wrong side (left or right) of the target object. Here, we only investigate trajectories that start on the wrong side, that is, if the end-effector has to approach the object from the right side, it is started on the left side of the target. Thus, none of the approaches can achieve close to zero percentage. As can be seen, all LTO versions of our approach achieve excellent results, even outperforming the human demonstrations. Furthermore, the high percentage of CHOMP illustrates that the task does require reasoning beyond obstacle avoidance.

Move-Can: On this task, the trend in collision avoidance performance is similar to the one on the previous task. The end-effector's deviation from the vertical equally confirms our finding that simply using points on the discrete graph outperforms CHOMP on the task specific measure, but can be significantly improved via local trajectory optimization. Again, all locally optimized trajectories using our learned cost function are able to outperform the human demonstration, holding the cup more consistently upright than the person.

Laptop: In the most complex task, which requires holding a cup upright and not moving it above electronic devices, the human and the LTO approaches still achieve excellent obstacle avoidance, while the discrete graph and CHOMP generate more collisions due to the complexity of the scenes. As in the Move-Can task, our learned trajectories outperform the human in the ability to hold the cup upright during motion. Furthermore, the large normal deviation of CHOMP indicates that the good performance of our approach is not only due to the setup of the task, but due to specific aspects of the cost function learned from demonstrations. The third column of the Laptop experiment shows the percentage of trajectory points above the laptop. Even here, the best version of our approach slightly outperforms the human and significantly improves on the discrete graph paths. Again, the large values for CHOMP show that the laptop had to be explicitly avoided through a learned cost function. A video showing our results with the PR2 can be found at [Byravan, 2015].

Locally-optimal IOC: To investigate whether local techniques are able to learn cost functions in our setting, we ran IOC directly on two versions of LTO, initializing LTO with a single straight line path and with multiple randomly sampled paths. For the latter, we averaged the feature expectations of the LTO outputs to compute the gradient. Both these algorithms failed to converge, resulting in poor cost function estimates. Another version of the algorithm that tried to refine the cost function learned from our graph-based method also failed to converge. This confirms our belief that local trajectory optimization techniques are not capable of learning generalizable cost functions in our problem setting, which involves complex cost functions and requires reasoning over multiple trajectories among obstacles. We thus expect similar issues with applying other locally-optimal methods such as [Levine and Koltun, 2012; Jain *et al.*, 2013].

4.4 Discussion

The key findings from our experiments are: (1) Our approach is able to learn complex cost functions from demonstrations, learning to trade off multiple objectives such as not colliding with objects while holding a cup upright and not moving above a laptop. (2) The weaker performance of CHOMP on the task specific metrics indicates that our test tasks do indeed require cost functions beyond obstacle avoidance. (3) While paths sampled from the discrete graph are too coarse to pro-

vide very good test performance, the graph representation is sufficient for learning cost functions that provide excellent results when used in combination with our local trajectory optimizer (LTO). (4) LTO achieves the overall best results when initialized with multiple paths sampled from the discrete graph according to the learned cost, often outperforming the human demonstrator on task specific measures. (5) LTO achieves almost equally good results when initialized with random paths, highlighting the strength and generalizability of our learned cost function, which provides good basins of attraction for trajectory optimization. This is an advantage, as we can cheaply generate random paths at test time, enabling near real-time operation. (6) Locally-optimal IOC using LTO fails to converge, indicating that purely local approaches are limited in their capacity to learn expressive cost functions and integration of more global information is necessary.

5 Conclusion

We presented an approach for Inverse Optimal Control that is able to learn cost functions for manipulation tasks. Our approach performs IOC in a discrete, graph-based state space and further refines trajectories using local trajectory optimization on the learned cost function. The discrete graph representation has several advantages, including the ability to reason about arbitrary distributions over paths, flexibility in the feature representation, and better theoretical learning guarantees. We showed results from testing the approach on three manipulation tasks with two 7DOF robots. By comparing the approach with human demonstrations on held out data, we showed that the algorithm is able to learn meaningful behaviors that match or exceed those from the demonstrations.

Despite these very promising results, there are several areas that warrant further research. One limitation of our current approach is due to the sampling strategy, which requires re-generation of the graph for every task and might not scale to even higher dimensional planning problems. Learning better sampling strategies from training experiences might enable the use of smaller graphs, or graphs that don't have to be re-generated for each task. A hierarchical sampling approach might further increase the efficiency of the graph representation. We already started investigating the use of optimized GPU implementations for efficient feature computation, and we believe that it will be possible to scale the approach to dynamic settings, where the manipulator has to constantly replan its path among moving obstacles and people. Such an extension would enable our learning technique to be deployed in scenarios involving real-time interactions with people.

6 Acknowledgments

This work was funded in part by the National Science Foundation under contract NSR-NRI 1227234 and Grant No: 1227495, Purposeful Prediction: Co-robot Interaction via Understanding Intent and Goals.

References

[Abbeel and Ng, 2004] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.

[Aghasadeghi and Bretl, 2011] Navid Aghasadeghi and Timothy Bretl. Maximum entropy inverse reinforcement learning in continuous state spaces with path integrals. In *IROS*, 2011.

[Byravan, 2015] Arunkumar Byravan. Project website: Graph-based IOC. `http://rse-lab.cs.washington.edu/projects/graph-based-ioc`, 2015.

[Curless, 1997] Brian Curless. *New Methods for Surface Reconstruction from Range Images*. PhD thesis, Stanford University, 1997.

[Dragan *et al.*, 2011] Anca Dragan, Geoffrey J Gordon, and Siddhartha Srinivasa. Learning from experience in manipulation planning: Setting the right goals. In *ISRR*, 2011.

[Grünwald and Dawid, 2004] Peter D. Grünwald and A. Phillip Dawid. Game theory, maximum entropy, minimum discrepancy, and robust Bayesian decision theory. *Annals of Statistics*, 2004.

[Henry *et al.*, 2010] Peter Henry, Christian Vollmer, Brian Ferris, and Dieter Fox. Learning to navigate through crowded environments. In *ICRA*, 2010.

[Jain *et al.*, 2013] Ashesh Jain, Brian Wojcik, Thorsten Joachims, and Ashutosh Saxena. Learning trajectory preferences for manipulators via iterative improvement. In *NIPS*, 2013.

[Kalakrishnan *et al.*, 2011] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *ICRA*, 2011.

[Kalakrishnan *et al.*, 2013] Mrinal Kalakrishnan, Peter Pastor, Ludovic Righetti, and Stefan Schaal. Learning objective functions for manipulation. In *ICRA*, 2013.

[Levine and Koltun, 2012] Sergey Levine and Vladlen Koltun. Continuous inverse optimal control with locally optimal examples. In *ICML*, 2012.

[Ng and Russell, 2000] Andrew Y Ng and Stuart J Russell. Algorithms for inverse reinforcement learning. In *ICML*, 2000.

[Ratliff *et al.*, 2009] Nathan D Ratliff, David Silver, and J Andrew Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 2009.

[Schulman *et al.*, 2013] John Schulman, Alex Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. *RSS*, 2013.

[Topsøe, 1979] Flemming Topsøe. Information theoretical optimization techniques. *Kybernetika*, 1979.

[Ziebart *et al.*, 2008] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, 2008.

[Ziebart *et al.*, 2012] Brian Ziebart, Anind Dey, and J Andrew Bagnell. Probabilistic pointing target prediction via inverse optimal control. In *IUI*, 2012.

[Ziebart, 2010] Brian D Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. PhD thesis, Carnegie Mellon University, 2010.

[Zucker *et al.*, 2013] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. CHOMP: Covariant hamiltonian optimization for motion planning. *IJRR*, 2013.