Learning from Physical Human Feedback: An Object-Centric One-Shot Adaptation Method

Alvin Shek

August 9, 2022

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Changliu Liu
Oliver Kroemer
Ruixuan Liu

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

To my family.

Abstract

For robots to be effectively deployed in novel environments and tasks, they must be able to understand the feedback expressed by humans during intervention. This can either correct undesirable behavior or indicate additional preferences. Existing methods either require repeated episodes of interactions or assume prior known reward features, which is data-inefficient and can hardly transfer to new tasks. We relax these assumptions by describing human tasks in terms of object-centric sub-tasks and interpreting physical interventions in *relation to specific objects*. Our method, Object Preference Adaptation (OPA), is composed of two key stages: 1) pre-training a base policy to produce a wide variety of behaviors, and 2) online-updating only certain weights in the model according to human feedback. The key to our fast, yet simple adaptation is that general interaction dynamics between agents and objects are fixed, and only object-specific preferences are updated. Our adaptation occurs online, requires only one human intervention (one-shot), and produces new behaviors never seen during training. Trained on cheap synthetic data instead of expensive human demonstrations, our policy correctly adapts to human perturbations on realistic tasks in both simulation and on a physical 7DOF robot.

Acknowledgments

I would like to thank my advisor Professor Changliu Liu and my mentor Rui Chen for their advice and guidance throughout my Master's study. From sorting through much existing work to understanding how to even conduct research, the skills I have learned are thanks to their great mentorship and patience. This thesis would not have been possible without them.

I would also like to thank Professor Oliver Kroemer and Ruixuan Liu for being my committee members. They have provided meaningful feedback and insights during our meetings.

Last but not least, I am forever indebted to my family and relatives for supporting my decisions and giving me the opportunity to pursue my own interests. I am also thankful for the amazing friends I have made at the Robotics Institute.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robots are useful in many real world tasks, e.g., where there are health risks or where tedious efforts are required. Ways to achieve desired behavior range from traditional planning and control to reinforcement or imitation learning. Cost functions, rewards, and pre-recorded demonstrations that capture the target behavior are often fixed beforehand. However, during actual deployment, unpredictable factors can always arise, such as both varying task specifications as well as human preferences on how those tasks should be carried out. Such changes can be easily and naturally handled if a human can provide corrective physical feedback in real time. In such physical human-robot interactions (pHRI), robots need to understand this feedback and adapt their behavior accordingly for future tasks. In this paper, we study a special yet prevalent setting where the online human feedback is *object-centric*, as opposed to non-object-centric settings such as those encoding temporal preferences. That is, human interventions can be interpreted as leading the robot to either avoid or visit certain objects and areas in the environment with possibly preferred orientations. Hence, by leveraging this assumption, the robot can infer which objects are relevant and better understand human feedback.

Illustrating Example To better understand this insight, consider a factory setting where a robot has been trained to carry printed items from a 3D printer to a bin for cleaning. Now, the manufacturing process suddenly changes: the robot should additionally place items upright in a scanner before dropping them in the cleaning bin.

As a human operator, a natural and quick way to communicate this task modification would be a physical correction. After the robot picks up an item, the human would drag and guide the arm towards the scanner and have the end-effector hold the item upright for scanning before guiding the arm towards the bin. This detour could be simply described as an additional object-centric sub-task, which requires the item to be moved with an upright orientation *relative to* the downward-facing scanner. In this work, we take advantage of this object-centric interpretation of human intentions to enable fast, yet simple online adaptation of robot behaviors.

Existing Approaches Several approaches have been proposed to adapt robot behaviors using human feedback online. [Bajcsy et al., 2017] uses human perturbations to infer desired trajectories and incrementally updates learned preferences using maximum posteriori estimation (MAP). [Losey and O'Malley, 2019] assumes that humans intervene based on a linear combination of position and velocity error, and derives an update rule for learned preferences. Although these approaches enable fast, effective adaptation, they assume human preferences are weights for a linear combination of basis functions, such as distance to either a pre-specified object or relative orientation with respect to the goal. However, the linear assumption has limited expressiveness, and the pre-defined basis functions might not cover changing human preferences. This is especially problematic when robots are deployed with new tasks, such as scanning labels under a newly-installed scanner. Robots should be able to extract new and flexible preferences from human feedback and leverage them to better adapt future actions.

Other approaches avoid predefined features and rather rely on deep neural networks to learn these from human demonstrations. [Finn et al., 2016, Wulfmeier et al., 2016] use the raw state space in the form of RGB images as input and evaluate on tasks like pouring into a target cup, placing a dish into a dish rack, and even autonomous driving. As a consequence of such a high dimensional state space, these learned reward functions are constrained to test environments that match those available during training. Techniques such as transfer learning pretrained feature extractors [Bengio, 2012] or domain randomization Tobin et al. [2017] could possibly alleviate this issue. Our approach is robust to environment arrangement and number of objects through our object-centric, graph-based formulation.

[Bobu et al., 2021] enforce structure into human demonstrations by requiring that such trajectories move from a region of high reward to a region of low reward. They train a neural network to correctly compare pairs of states based on relative reward. [Bobu et al., 2022] relax this requirement of monotonically decreasing reward by allowing humans to pick the start and end reward values. By learning entirely new reward functions from scratch, however, these methods require multiple demonstrations to sufficiently remove ambiguity in the desired motion, thus preventing them from adapting online. [Mehta and Losey, 2022] takes a similar approach in learning new reward functions as MLP's from scratch; adaptation thus requires several samples of human feedback. They offer more flexibility in allowing demonstrations, physical corrections, and comparisons in one unified framework. Although training small, simple MLP's reduces the time and amount of data necessary to converge, such MLP's may not generalize well to different environments with different arrangements of the same objects. Moreover, MLP's have a fixed input size associated with a specific set of object states as input; it is not clear how to use previously trained MLP features when new objects are encountered and old ones disappear.

Our Approach To address the above challenges, we propose Object Preference Adaptation (OPA) to 1) explicitly consider pairwise relations between each object and the robot and 2) interpret human feedback in terms of these relations. We first pre-train a policy to reproduce randomly-generated trajectories that follow a specific pattern: reaching a goal while interacting with nearby objects. The policy parameters are divided into two groups: the core weights encoding the general interaction dynamics between objects, and the object-specific features. We assume there are only four possible classes of object interactions during data generation: either moving closer or farther away in position, and either ignoring or matching the orientation of an object with a fixed rotational offset. Real-world examples of this include moving above and horizontally tilting to pour water into a cup, or staying away from an open flame. As the policy learns to imitate these trajectories, it also learns two key properties: how objects generally influence a trajectory (the interaction dynamics), and how specific types of object relations can be represented in a continuous latent space of finite dimensions.

When adapting to physical perturbations, the core policy weights are frozen because the general dynamics of object interactions should not change. Rather, *object-specific features* need to be adapted to capture various behaviors. In the illustrating example, the scanner should always be able to attract or repel the robot (the dynamics), but which specific behavior and to what extent are object-specific. We argue that such separation can be naturally captured by a graph with the agent and objects as nodes. The interaction dynamics are shared among all edges, while how each node pair influences the robot behaviors also depends on the actual objects. Following this idea, we compose our robot policy based on the Graph Neural Network (GNN), which further allows end-to-end learning given human feedback. As a result, during online adaptation, we constrain the search to a compact latent space of object-specific features, which can be done with only a few gradient steps. Although our pre-training data contained only four classes of object relations, optimization over a continuous latent space allows the policy to express new, unseen types of interactions with objects, such as ignoring objects for position, or learning arbitrary 3D orientations relative to an object's orientation.

To evaluate our model's adaptability, we invited users to perturb a simulated robot manipulator in three separate tasks. Users would observe the robot's original behavior, apply perturbations to demonstrate desired behavior, and judge the robot's updated behaviors. We compare to a baseline which switches to gravity compensation during perturbation, but otherwise tracks the goal in a predefined way. Based on the user study, OPA shows better understanding of human preferences, requires less feedback, and responds to corrections more predictably. We also evaluate our method on a sequence of real-life, physical tasks using a 7DOF Kinova Robot and demonstrate adaptability to real, physical perturbations.

Overall, we summarize our contributions as follows:

1. We interpret human physical feedback as object-specific interactions and model these with a graph representation, enabling fast adaptation.
2. We show that synthesized data alone can train a policy that handles realistic tasks and unstructured human perturbations.
3. We experimentally show adaptability to human perturbation on three simulated tasks as well as one real robot task.

Chapter 2

Related Work

Just like how people communicate desired tasks to others, a common approach in robotics is to directly communicate desired behavior to robots through demonstrations. The correlation between demonstrated states and actions can then be learned in a supervised manner, a technique known as imitation learning [Osa et al., 2018]. What is not learned, however, is *why* such actions are desirable, which is important for generalizing to new scenarios with sufficiently different state distribution [Abbeel and Ng, 2004]. Inverse Reinforcement Learning (IRL) attempts to solve this by instead learning a reward function that captures the desired human behavior [Osa et al., 2018]. The human's actions presumably maximize this reward function parameterized by θ, and the goal is that inferring correct human preferences θ can help model the correct robot behavior.

2.1 Inverse Reinforcement Learning

IRL has been applied to more than just demonstrations [Abbeel and Ng, 2004]; other forms of feedback include physical corrections [Bajcsy et al., 2017, Jain et al., 2016, Losey and O'Malley, 2019], rankings [Brown et al., 2019b], and pairwise comparisons [Christiano et al., 2017]. These works also define the interaction between humans and robots differently. [Jain et al., 2016, Losey and O'Malley, 2018] perform multiple, iterative rounds of human feedback and robot learning, whereas [Brown et al., 2019a, Lopes et al., 2009] have the robot actively query the human to maximize information gain. [Bajcsy et al., 2017, Losey and O'Malley, 2019] handle human physical feedback by adapting online; our work takes this approach to minimize teaching effort from the human.

2.2 Hand Crafted Features

An important aspect separating these works is how their reward functions are defined, and specifically what state space they take as input. Given that the world can be complex to model, a state space must be chosen that both generalizes well to different scenarios and enables reasonable computational time. Observations are also often limited and noisy, which creates much ambiguity: many reward functions could represent one set of human inputs [Ziebart et al., 2008]. IRL methods commonly address this by constraining the space of rewards to be composed of pre-specified basis functions $\phi(x)$ with unknown weights θ: $r(x) = \theta^T \phi(x)$ [Abbeel and Ng, 2004, Bajcsy et al., 2017, Jain et al., 2016, Losey and O'Malley, 2019]. Although optimization of a convex loss with respect to these weights θ is convex, the chosen features $\phi(x)$ need to be specified apriori by an expert. In certain applications, these features are indeed known and fixed, such as holding a coffee cup upright, but in rapidly changing environments like the household, this assumption may not hold. Optimizing over the wrong features may not fully capture the human's desired behavior [Bobu et al., 2018, Haug et al., 2018], leading to incorrect robot behavior.

2.3 Learning Features Directly from State Space

Recent works have attempted to address this by indirectly learning such features through deep neural networks [Brown et al., 2020, Finn et al., 2016, Mehta and Losey, 2022, Wulfmeier et al., 2016]. These deep IRL methods take in the raw state space as input and rely on the networks to learn a good representation that transfers to test time scenarios. One challenge, however, is that given a limited set of demonstrations and a high-dimensional state space, deep neural networks may learn features that do not generalize to new scenarios [Fu et al., 2017]. The higher dimensional the state, the more diverse the data needs to be reduce ambiguity in the humans' true preferences.

[Bobu et al., 2021, 2022] tackle this challenge by adding more structure into the human demonstrations themselves: trajectories should move from a region of high reward to one with low reward. Reward can be assumed to monotonically decrease from start to end for each trajectory, allowing for any pair of states along the trajectory to be compared and thus producing a large amount of data. Using a Boltzmann model that assumes human corrections are noisy but rational, they can then incrementally learn new feature functions as separate neural networks when a new human correction cannot be confidently modelled. To reduce ambiguity in state space, however, the human needs to provide multiple different trajectories to teach new behavior. [Mehta and Losey, 2022] follows a similar strategy of learning new neural network-based reward functions from scratch. Rather than compare individual pairs of states along a trajectory, they perturb a human demonstration with randomly sampled noise, and optimize the new network to assign highest reward to the original human trajectory compared to the other artificial ones. Both these works do not necessarily rely on but instead allow for hand-crafted feature bases if available. The key question is whether or not these reward functions trained from scratch can sufficiently generalize to new scenarios. Training time is another concern, as [Mehta and Losey, 2022] and [Bobu et al., 2021] require around 2 minutes to learn a new behavior, not including the human interaction time.

2.4 Object-Centric Manipulation Policies

The idea of representing the world as a graph of object-based nodes has been applied to modeling particle physics [Battaglia et al., 2016, Sanchez-Gonzalez et al., 2020], complex robot dynamics [Sanchez-Gonzalez et al., 2018], and even visual scenes [Agarwal et al., 2020]. Existing methods also apply this to visually imitate demonstrations [Sieb et al., 2020] and hierarchically compose object-centric controllers [Sharma et al., 2020]. Graphs can decompose complex dynamics and behaviors into a collection of simpler, object-object interactions. This has served as important inductive bias for helping Graph Neural Networks (GNN) generalize beyond training data [Battaglia et al., 2018]. We extend this idea to IRL and the challenge of understanding human preferences from unstructured, physical perturbations.

2.5 System Identification and Adaptation

Our method pre-trains a base policy and only adapts object-specific features at test-time to dramatically change behavior. This is similar to methods that condition a policy on real physics parameters [Yu et al., 2017] or latent embeddings [Kumar et al., 2021]. Rather than estimating the parameters of the world [Kumar et al., 2021], we estimate the object preferences of a human. Since we have some form of supervision through human perturbations, our latent features can be adapted using standard gradient descent as opposed to learning an explicit adaptor network [Kumar et al., 2021]. By separating latent features by object, our latent feature space is much more compact and easier to optimize over.

Chapter 3

Problem Formulation

At time t, let $x_{r,t} \in \mathbb{R}^d$ represent the current state of the robot r, and $x_{i,t} \in \mathbb{R}^d$ represent the states of other objects $i \in [N]$ in a scene. This assumes we can detect, uniquely identify, and extract the 6D pose of each object in a scene. We leave unstructured environments for future work.

The robot's state updates according to dynamics $x_{r,t+1} = x_{r,t} + \alpha u_t$ where α is a constant step size and $u_t \in \mathbb{R}^{d'}$ is the action either applied externally by the human if available, or otherwise generated by the policy. We denote the human intervention as u_t^* and the robot policy as $\hat{u}_t = f_O(x_{r,t}, \{x_{i,t}\}_{i\in[N]}, g)$ where the goal g is assumed given. The general learning objective is to match the human's actions over T steps:

$$\min_{f_O} \sum_{t\in[T]^*} \|f_O(x_{r,t}, \{x_{i,t}\}_{i\in[N]}, g) - u_t^*\|, \tag{3.1}$$

where $[T]^*$ is the set of time indices where human interventions are available.

Now, what form should f_O take? As we discussed in Section 1, existing works either focus on fast, online adaptation or try to replace predefined feature functions with learned MLP's. Training networks from scratch, however, requires enough data to reduce ambiguity in the state space, and this conflicts with reducing human effort in teaching robots. To ensure fast training with minimal data, such MLP's must be small and simple and thus may not generalize to different object arrangements. MLP's also have a fixed input size that becomes problematic when objects appear or disappear. These issues motivate us to represent the scene as a graph that can flexibly handle different numbers of objects.

In our graph, nodes are defined by the objects in the scene, and directed edges point only from each object to the robot. This strict definition of edges means that we only model interactions between the robot and each object, but not amongst the objects themselves. This practical assumption makes sense for most tasks that don't involve many dynamically interacting entities, and leaves us with several benefits. First, each object's influence on robot behavior can now be isolated, which also allows adaptation of human preferences to be isolated. Second, computation and overall network learning is simplified greatly with the sparse graph, allowing run-time to scale only linearly with the number of objects. Third, a graph in general allows objects to be easily added or removed without changing the fundamental computation.

In our graph, each of the agent-object relations is composed of two elements: relative state features $b_{i,t}$ and learned "preference" features c_i. State features are computed from robot, object, and goal state $b_{i,t} = b(x_{r,t}, x_{i,t}, g)$ and help the policy reason about spatial proximity and direction of each object relative to the robot and its goal. These should be generic and fast to compute while relieving burden on the network needing to learn these. Preference features c_i capture the specific way that each object should interact with the agent, and should be learned jointly with the rest of the network. Together, these can be processed by a *relation network* f_r to generate latent edge features $e_{i,t} = f_r(b_{i,t}, c_{i,t})$, as shown in the top of Fig. 4.1. We can rewrite the robot's overall policy as an "aggregator" f_O of edge features $f_O(\{e_{i,t}\}_{i\in[N]})$ and its objective as

$$\min_{\{c_i, f_r\}} \sum_{t\in[T]^*} \|f_O(\{e_{i,t}\}_{i\in[N]}) - u_t^*\|. \tag{3.2}$$

It is important to emphasize that relation network f_r is agnostic to the input robot-object edge, which is why new objects can be easily handled without any major fine-tuning. The aggregator f_O is what allows the graph network to handle an arbitrary number of edges and nodes. We also emphasize that preference features c_i are learned while the state features $b_{i,t}$ are fixed. This naturally isolates the objective information of the scene from subjective preferences and helps the model to focus on object-specific preference features.

Chapter 4

Approach

4.1 Object-Centric Representation

We now consider the specific domain of full rigid body motion in 2D or 3D for OPA. Time index t is omitted for clarity. We assume object-specific tasks can generally be represented as reach and avoid, and only focus on this setting without considering the complexity of actual grasping. We leave this for future work. The objects and robot are represented as spheres with radii of influence $s_{i \in [N]}$ and s_r respectively. Tricks to handle non-spherical objects are discussed in task 1 of the experiments.

Actions u are composed of translation v and rotation w actions. v is a unit vector describing the direction of translation. w describes the desired orientation at the next timestep. For 2D, $w = [\cos(\theta), \sin(\theta)]$ for angle θ on the x-y plane. For 3D, $w = [R_x^T, R_y^T] \in \mathbb{R}^6$ where $R_x, R_y \in \mathbb{R}^3$ are the x and y axes of an orientation expressed by a rotation matrix $R = [R_x, R_y, R_z] \in SO(3)$. This is shown in the top right of Fig. 4.1, where $\hat{v}_t$ points from the end-effector's current to future position, whereas $\hat{w}_t$ is the future orientation.

States x_i are separated by position $x_{P,i}$ and orientation $x_{R,i}$. State features b_i are separated by position $b_{P,i} = b_P(x_r, x_i, g)$ and orientation $b_{R,i} = b_R(x_r, x_i, g)$. Preference features c_i are composed of position component $c_{P,i}$ and orientation component $c_{R,i}$. $c_{P,i}$ is purely latent whereas $c_{R,i} = [c_{R,i}^{\text{latent}}, c_{R,i}^{\Delta}]$ is composed of a latent part $c_{R,i}^{\text{latent}}$ and a learned rotational offset $c_{R,i}^{\Delta}$. This learned offset is applied directly to the input orientation of an object and allows the model to learn relative orientations. For example, scanning the barcode underneath a bowl in Fig. 4.1 requires the barcode to face the scanner, and this relative orientation can be represented by a rotational offset.

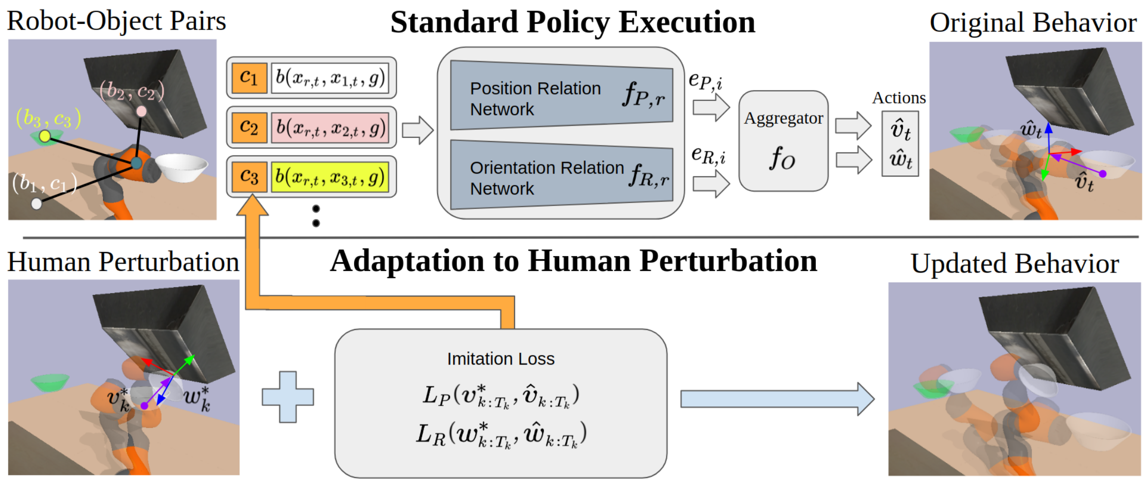
Robot-Object Pairs
Standard Policy Execution
Original Behavior
(b_2, c_2)
(b_3, c_3)
(b_1, c_1)
c_1 $b(x_{r,t}, x_{1,t}, g)$
c_2 $b(x_{r,t}, x_{2,t}, g)$
c_3 $b(x_{r,t}, x_{3,t}, g)$
Position Relation Network $f_{P,r}$
Orientation Relation Network $f_{R,r}$
$e_{P,i}$
$e_{R,i}$
Aggregator f_O
Actions $\hat{v}_t$ $\hat{w}_t$
$\hat{w}_t$
$\hat{v}_t$
Human Perturbation
Adaptation to Human Perturbation
Updated Behavior
v_k^*
w_k^*
Imitation Loss
$L_P(v^*_{k:T_k}, \hat{v}_{k:T_k})$
$L_R(w^*_{k:T_k}, \hat{w}_{k:T_k})$

Figure 4.1: Illustration of the adaptation with OPA. OPA consists of a position relation network $f_{P,r}$ and an orientation relation network $f_{R,r}$ whose outputs are aggregated by f_O. **Top:** For each object $i \in [N]$ in a scene (including the green goal), a state-based feature $b_R(x_{r,t}, x_{i,t}, g)$ is computed from both object state $x_{i,t}$ and robot state $x_{r,t}$. This is passed in along with a learned, preference feature c_i into each relation network to produce edge features $e_{P,i}$ and $e_{R,i}$. Together, all edge features are fed into an aggregator function f_O that finally computes predicted translation $\hat{v}_t$ shown in purple and orientation $\hat{w}_t$ shown as Cartesian axes in the top right. **Bottom:** In the bottom left, humans can perturb both position and orientation to produce $v^*_{k:T_k}$ and $w^*_{k:T_k}$, which can be treated as ground truth for an imitation loss. Only object preference features c_i are updated, allowing for fast and effective adaptation of behavior such as scanning the bottom of a bowl as shown in the final image to the right.

4.2 Graph Representation Overview

To process a graph of arbitrarily many object nodes, we choose to use a GNN due to its desirable property of invariance to both the number and ordering of nodes. Also, since GNN's apply the same network to process every edge in a graph, they enforce a strong inductive bias that general edge computations remain the same. Only each pair of node's relative features distinguishes specific behavior. This is the key for GNN's well-known generalization to unseen scenarios.

To process each robot-object node pair, two different networks compute latent edge features: $e_{P,i} = f_{P,r}(b_{P,i}, c_{P,i})$ for position, and $e_{R,i} = f_{R,r}(b_{R,i}, c_{R,i})$ for orientation. Finally, the output actions can be computed using the same function f_O but with different sets of edge features: $\hat{v} = f_O(\{e_{P,i}\}_{i\in[N]})$ and $\hat{w} = f_O(\{e_{R,i}\}_{i\in[N]})$, as shown in Fig. 4.1. In the following sections, we will explain the computation of these features for each object and how these are aggregated to produce a single output action.

4.3 Relation Network and Aggregator

In this section, we first discuss our general relation network f_r used for both position and orientation control, which takes as input pairs of state and preference features (b_i, c_i).

Typically, neural network inputs are raw states, hand-crafted state features, or outputs from a separate network. Our approach, however, treats the object-specific features c_i as updatable weights to be learned with the rest of the network through gradient descent. As our policy trains on the same set of object types, these low-dimensional preference features gradually discriminate to represent very different behaviors. In fact, as will be described in later sections, these object feature spaces can be as small as 1D and still capture diverse behaviors. We note that during training, the object types are provided as input in the form of indices. This enables the model to associate the correct feature c_i with each object, ensuring that they are trained consistently.

Pairs of state-based and preference features are fed into a multi-layer perceptron (MLP) f_r^1 to produce intermediate features $\tilde{e}_i$. A second MLP f_r^2 computes unnormalized attention-like weights $\tilde{\alpha}_i$ for these features, and overall edge features $e_i = \tilde{\alpha}_i \tilde{e}_i$ take the form of actual actions. Given this set of weighted actions, the aggregator f_O simply performs a summation followed by a normalization.

For position action $\hat{v}$, we normalize its magnitude to produce a valid unit vector:

$$\hat{v} = \frac{\sum_{i\in[N]} e_{P,i}}{||\sum_{i\in[N]} e_{P,i}||} \tag{4.1}$$

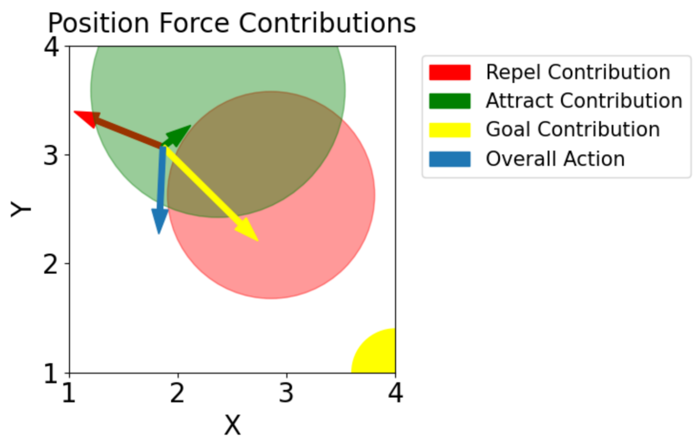

Figure 4.2: Position "force" contribution $e_{P,i}$ of each object node along with overall action $\hat{v}$ for a 2D synthetic scene.

Each object essentially contributes a "force" pushing or pulling on the robot, as shown in Fig. 4.2. For orientation $\hat{w}_t$, we sum up all edge features and normalize for each specific axis:

$$\tilde{w} = \sum_{i \in [N]} e_{R,i} \text{ then } \tilde{w} = \begin{bmatrix} \frac{\tilde{w}_{1:3}}{||\tilde{w}_{1:3}||} & \frac{\tilde{w}_{4:6}}{||\tilde{w}_{4:6}||} \end{bmatrix}. \tag{4.2}$$

For the 2D case, $\hat{w} = \tilde{w} \in \mathbb{R}^2$. For the 3D case, however, since our above calculation is essentially a normalized weighted sum of rotation matrices, the resulting $\tilde{w} = [\tilde{R}_x^T, \tilde{R}_y^T]$ is not a valid rotation. $\tilde{R}_x$ and $\tilde{R}_y$ must be orthogonal, and to enforce this, we apply Gram-Schmidt orthogonalization to remove the component of $\tilde{R}_y$ parallel to $\tilde{R}_x$ while keeping $R_x = \tilde{R}_x$:

$$\tilde{R}'_y = \tilde{R}_y - \frac{\langle \tilde{R}_x, \tilde{R}_y \rangle}{\langle \tilde{R}_x, \tilde{R}_x \rangle} \tilde{R}_x \text{ and } R_y = \frac{\tilde{R}'_y}{||\tilde{R}'_y||}. \tag{4.3}$$

The resulting two axes $\hat{w} = [R_x^T, R_y^T]$ alone are sufficient to represent a full rotation in $SO(3)$, where the R_z axis is simply the cross product of the R_x and R_y axes.

4.4 Position Relation Network

In this section, we describe state-based and preference features for our position relation network. State-based features $b_{P,i}$ are the following:

1. **Size-relative distance**: Distance between robot and object divided by the sum of their radii, which helps the policy reason about when to interact with an object.
2. **Direction**: Unit-vector pointing from agent to each object, which helps determine the direction of "force" applied on agent.
3. **Goal-relative direction**: Inner-product between each agent-object vector and the agent-goal vector. A positive value indicates that the object lies in the same direction as towards the goal and should be considered. A negative value indicates the object is "behind" the agent and can be ignored.

Position preference features $c_{P,i}$ intuitively capture the magnitude of the output force, or how attracted or repelled the robot should be from each object. Reducing its dimensionality as much as possible prevents over-fitting while simplifying adaptation in the feature space, and in our experiments, simply using 1D was enough for expressive policy behavior.

The position relation network overall outputs a push-pull force on the agent. Potential field methods [Latombe, 1991] also use this approach, but constrain this force vector to be parallel to each agent-object vector. This may seem like an intuitive way to enforce structure in the network and reduce complexity, but this constraint fails during "singularities" where no orthogonal component is available to avoid an obstacle lying in the same direction as the goal.

During training, we measure misalignment between ground-truth v_t^* and predicted $\hat{v}_t$ translation directions using the following loss across B batch samples:

$$L_P = \frac{1}{B} \sum_{b=1}^{B} 1 - \langle v_b^*, \hat{v}_b \rangle \tag{4.4}$$

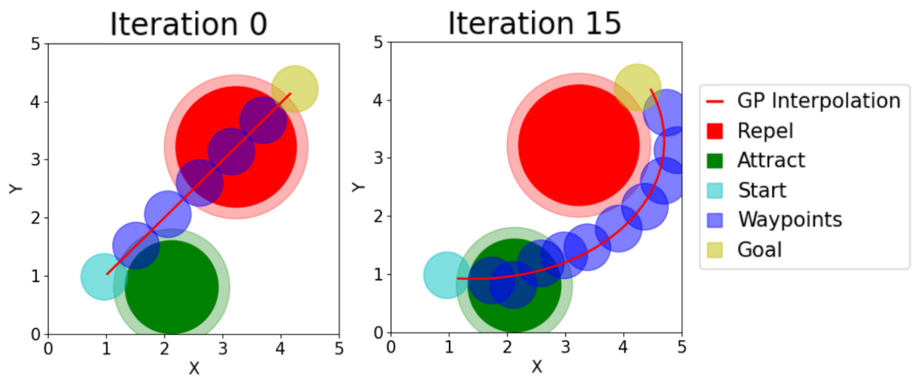

Figure 4.3: First and last steps of Elastic Band in 2D. Dark blue agent waypoints traverse from the light-blue start to yellow goal position while interacting with nearby "Attract" and "Repel" objects. A Gaussian Process interpolates between the waypoints.

4.5 Orientation Relation Network

In this section, we describe state-based and learned features for our orientation relation network. State-based features $b_{R,i}$ are the following:

1. **Size-relative distance**: identical to that of the position relation network.
2. **Modified Orientation**: Orientation of each object, but rotated by a learned rotational offset. This helps the policy output the correct orientation relative to an object's.

The learned rotational offset $c^{\Delta}_{R,i}$ is represented as a scalar $\Delta\theta$ for 2D and as a unit-normalized quaternion for 3D. $c^{\Delta}_{R,i}$ is applied to the x and y axes of the object's rotation matrix. We choose to manually apply a learned rotational offset to reduce burden on the network of needing to learn how to apply valid rotations. An additional learned feature $c_{R,i}$ intuitively determines whether an object's orientation should be ignored or not. This is important for modeling real-world tasks where a robot should move closer to an object without changing its orientation, such as handing a glass of water to a person.

Loss can be calculated by measuring the summed misalignment between ground-

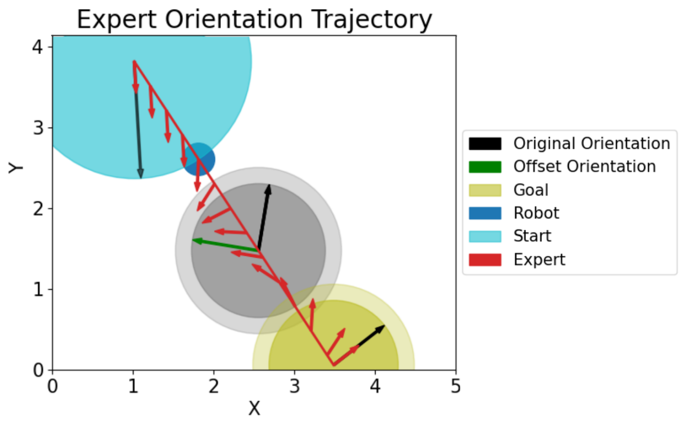

Figure 4.4: Example scenario to train the orientation relation network in 2D. Expert path in red is annotated with an orientation vector at each waypoint. Each node (start, grey object, and goal) has both its observed orientation $x_{R,i}$ shown as a black arrow along with its desired, relative orientation $R(c^{\Delta*}_{R,i}) \cdot x_{R,i}$ shown in green. The only node with nonzero rotational offset is the grey object: $c^{\Delta*}_{R,0} = -90°$, which is why the green arrows are not visible for start and goal.

truth and predicted x and y axes:

$$L_R = \frac{1}{B} \sum_{b=1}^{B} 2 - \langle R^*_{x,b}, \hat{R}_{x,b} \rangle - \langle R^*_{y,b}, \hat{R}_{y,b} \rangle. \tag{4.5}$$

4.6 Training and Online Adaptation

So far, we have discussed the implementation and intuition behind our policy. We now discuss how to train and adapt such a policy at test time.

4.7 Training with Synthetic Data

Imitation learning typically requires demonstrations, but collecting human demonstrations can require much effort [Mandlekar et al., 2019]. However, since our intended real-world tasks can be considered as reaching a goal with midway object interactions, we can generate synthetic data to capture this behavior. For position and orientation specifically, notice how their computation is completely independent. This allows us optimize their losses L_P and L_R separately, bypassing the issue where their units are different and thus require re-weighting [Groenendijk et al., 2020]. This also allows us to train both networks with different sets of data, which is necessary since the behavior of the position and orientation networks should not be correlated. For example, a trajectory may avoid an obstacle and stay far enough that the obstacle has no influence on the robot's orientation. By training orientation features on this data, the orientation network would learn to just ignore this object's orientation.

For the position network specifically, interactions with nearby objects should take the form of attractions and repulsions; Elastic Bands [Quinlan and Khatib, 1993] naturally model this. Fig. 4.3 shows this process in detail, where the final iteration's trajectory is used to train our position network.

For the orientation network, trajectories involve interactions with a single object of two possible relations: *ignore* and *consider*. Ignored objects have no influence on the agent's orientation. Considered objects force the agent to match their original orientation $x_{R,0}$ relative to an offset $c^{\Delta *}_{R,0}$ shared among all considered objects. Fig. 4.4 shows an example where the expert orientation of a waypoint must match that of nearby objects. The expert initially matches the start orientation with zero offset, but switches to the grey object's orientation in green with non-zero offset, and finally converges to the goal orientation with zero offset. Our policy would need to predict this relative orientation given only the original black orientation as input.

Since observed orientations $x_{R,0}$ are randomized while $c^{\Delta *}_{R,0}$ is fixed, our orientation network is forced to properly learn offset $c^{\Delta}_{R,0}$ to predict orientations $R(c^{\Delta *}_{R,0}) \cdot x_{R,0}$. Fig. 4.5 compares learned and true rotational offsets during training for the 2D case. The fact that they match shows that carefully chosen model structure can force the learned network weights to be interpretable. Another example in 3D is that, when pouring water from cup A into cup B with upright orientation $x_{R,B}$, we need to tilt cup A horizontally by $R(c^{\Delta *}_{R,B})$ to match the goal orientation $R(c^{\Delta *}_{R,B}) \cdot x_{R,B}$.

Our method relies on objects to influence behavior, but we also may desire the robot to fix its orientation throughout a trajectory, even with no objects nearby. A cup of water, for example, should be carried upright. To train this soft constraint, start and goal are treated as imaginary objects whose orientation must also be considered. Both share a true rotational offset of the identity, or zero: $c^{\Delta *}_{R,g} = R(0)$. Fig. 4.5 visualizes this in blue.

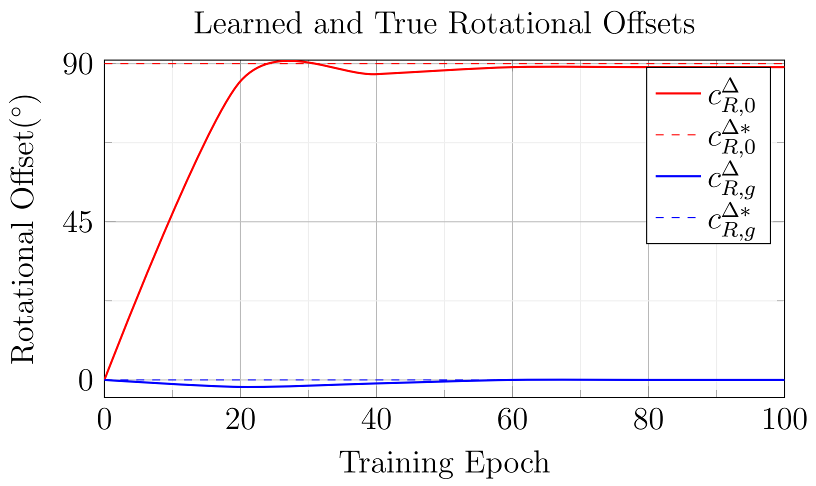

Figure 4.5: Comparing the ground truth $c^{\Delta*}_{R,i}$ and learned $c^{\Delta}_{R,i}$ 2D rotational offsets for the considered object $i = 0$ and goal $i = g$ during training. Here, $c^{\Delta*}_{R,0} = 90°$ and $c^{\Delta*}_{R,g} = 0°$.

4.8 Online Adaptation

In this section, we discuss our assumptions on the form of human perturbation as well as how the policy adapts to this to infer human preferences. People think and act differently, and this changes how and when they may intervene in the robot's trajectory. [Spencer et al., 2020] argues that people may intervene only when recent behavior has been unacceptable. This implies that the overall trajectory is not an example of a good trajectory, but rather a *transition* from bad to good. This thus provides information for *how the robot's behavior should change* rather than the absolute trajectory that should be naively imitated.

Following that intuition, we focus only on the human perturbation trajectory $(x^*_{k:T_k}, u^*_{k:T_k})$ and treat this as an expert trajectory to imitate. A key assumption is that humans only care about the final pose rather than the actual trajectory taken. Based on this assumption, we only take the start and end pose of the perturbation trajectory and linearly interpolate between the two. On the policy side, rather than calculating the output action for each individual expert state $x^*_{k:T_k}$, we rollout the policy in an open-loop fashion, starting from initial intervention state x^*_k and repeatedly taking the policy's action with deterministic dynamics. This "imagined" rollout makes two assumptions: the physical robot can perfectly track its desired pose, and objects' poses are fixed or can be predicted. Overall, however, this allows gradients to propagate throughout the entire rollout, and early mistakes will be penalized for future errors. Losses L_P and L_R from (4.4) and (4.5) can be calculated and used in standard gradient descent.

However, fine-tuning all weights of a neural network is well-known to lead to inefficient adaptation due to the large number of parameters. We bypass this issue by taking advantage of our graph-based architecture: updating only learned object features $c_{P,i}$ and $c_{R,i}$ while keeping the core relation network weights frozen. This matches our intuition: the general dynamics of interacting with objects should not change. Rather, *only object-specific features* need to be adapted to capture object-specific behaviors. This allows us to drastically change the policy's behavior with only a few steps of standard gradient descent.

This works surprisingly well, even for the learned rotational offset features $c^{\Delta}_{R,i}$. Recall that our model only had to learn two rotational offsets: $c^{\Delta *}_{R,0}$ and $c^{\Delta *}_{R,g}$. At test-time, however, our model can quickly adapt to reproduce *any arbitrary rotation* in $SO(3)$.

Chapter 5

Experiments

5.1 User Study

After pre-training, we evaluated our model's test-time adaptability to human perturbations. We invited 10 CMU students to participate in three simulated tasks where a robot manipulator's initial behavior was incorrect. Participants were instructed to press computer keys to perturb the end-effector's position or orientation at any moment. The current task would either continue or reset to show our model's updated behavior. The three tasks are shown in Fig. 5.3. Tasks 1 and 2 evaluate the position and orientation relation networks separately with certain features c_i initialized correctly. Task 3 evaluates both networks together in a more realistic scenario where no prior information about the object is known.

Task 1: Carrying Fragile Cup A robot bartender hands cups of water to customers. Since the cups are fragile, they should be carried low to the table. Participants would push the robot to stay close to and above the table when they felt necessary. The robot would either continue moving straight to the goal (baseline) or perform online adaptation and update its behavior (our model). An example of this updated behavior is shown in Fig. **??**. This scene contains three different table objects, and only their position preference features $c_{P,i}$ can be updated. This task highlights our model's fast online adaptation. Note that a flat table's position $x_{P,i}$ at time t is the orthogonal projection of the end-effector onto its surface.

Task 2: Inspecting Factory Items A robot presents pans to a quality control

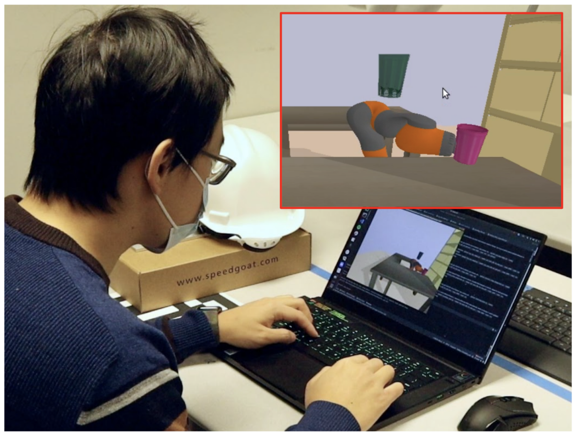

Figure 5.1: In task 1, the robot was perturbed as it moved towards the green floating cup representing the goal.

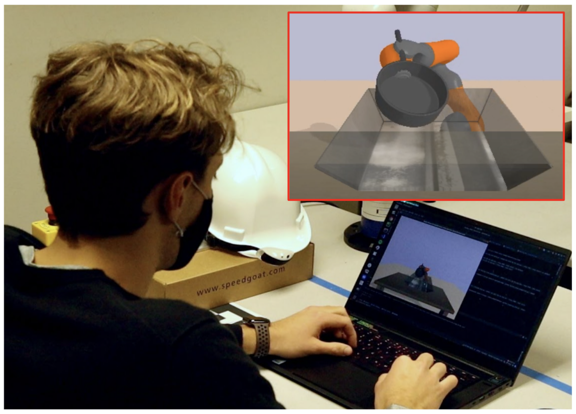

Figure 5.2: In task 2, the user rotated the end-effector to achieve a desired pan orientation above the inspection zone.

inspector who wants to view them at different orientations. Participants are shown an image of the pan's desired orientation and must perturb the robot-held pan to achieve this. Once satisfied, users then reset the episode and judge how closely the robot's updated orientation matches their final perturbed orientation (not the reference). This challenge involves only updates to the relative rotational offset $c^{\Delta}_{R,i}$ of the inspection bin, and evaluates the model's ability to represent arbitrary orientations in $SO(3)$. The linked video visualizes this clearly.

Task 3: Scanning Factory Items A robot initially carries manufactured bowls to a bin. The factory has installed a new barcode scanner, and the robot should midway scan the bottom of each bowl. Participants must perturb both position

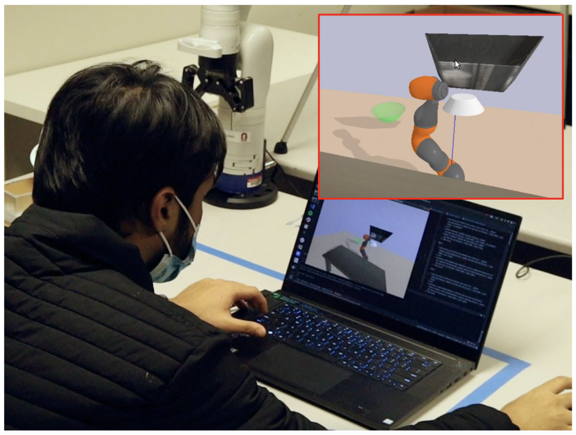

Figure 5.3: In task 3, the user perturbed the robot's position and orientation to demonstrate the additional scanning task before moving to the green goal location.

and orientation, requiring updates to all preference features of the scanner: $c_{P,i}$ for moving closer to it, $c_{R,i}^{\text{latent}}$ for actually caring about its orientation, and $c_{R,i}^{\Delta}$ for the upside-down orientation relative to it. This task is also shown in Fig 4.1.

5.2 Hyper-Parameters

For our experiments, we pre-trained our model using Adam with a learning rate of $lr = 3e-4$ and a batch size of 16 trajectories with 32 random timesteps sampled from each. We synthetically generated 3000 trajectories for both position and orientation datasets, taking 4 minutes each with 9 parallel processes. We trained both position and orientation networks for 100 epochs, taking 12 minutes each on a standard laptop GPU core.

We defined the first MLP's $f_{P,r}^1$ and $f_{R,r}^1$ of the position and orientation networks as ReLU-activated linear layers with output sizes [64, 64, 64]. The attention-generating MLP's $f_{P,r}^2$ and $f_{R,r}^2$ had output sizes [16, 8, 1] with activations [ReLU, ReLU, Softmax]. Preference features $c_{P,i}$ and $c_{R,i}^{\text{latent}}$ were only 1-dimensional, whereas rotational offsets $c_{R,i}^{\Delta}$ were 4-dimensional quaternions.

During online adaptation, we fine-tuned our object features with Adam and a learning rate of $lr = 0.1$, and specifically ran 5 gradient updates for the position network and 20 for the orientation network, taking less than 0.5 seconds in total.

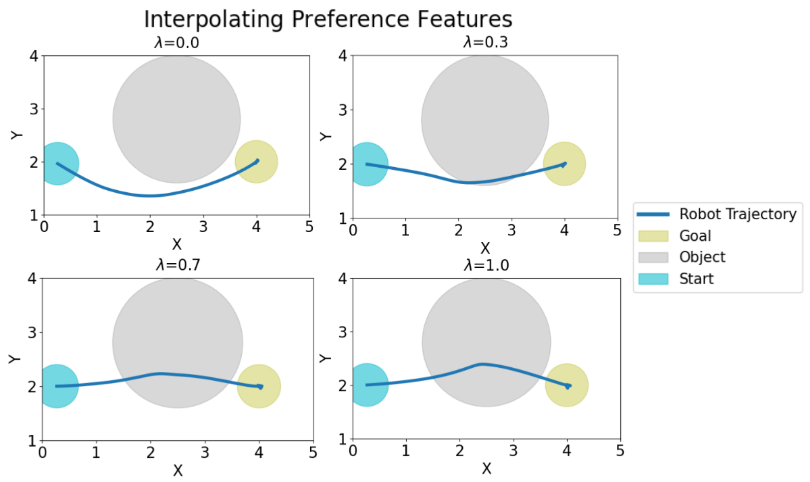

Figure 5.4: Resulting change in robot's behavior as the grey object's preference feature shifts between "repel" $c_{P,0}$ and "attract" $c_{P,1}$ according to $(1-\lambda)c_{P,0} + \lambda c_{P,1}$.

5.3 Feature Initialization

Optimizing deep neural networks is notoriously non-convex, so weight initialization is important to consider. During training, we specifically initialize position preference features $c_{P,i}$ of repulsion, attractor, and goal object as [1.0, 0.5, 0.0]. This prior tries to ensure that at the end of training, interpolating from one feature to another leads to interpolation in behavior space. This interpolation allows us to reasonably guess the feature value representing "ignore" behavior, which is how any new, unseen object should be initially treated. We visualize the interpolation in Fig. 5.4. As c_P of the grey object interpolates from learned "repel" $c_{P,0}$ to "attract" $c_{P,1}$, the robot clearly moves closer to the object. The indices 0 and 1 refer to an arbitrary repel object and an arbitrary attract object respectively. At test-time, any new objects can be approximately initialized as "ignore" with value $(c_{P,0} + c_{P,1})/2$.

For learned rotation offsets, especially in SO(3), we cannot rely on a fixed initialization due to saddle points and local optima when optimizing over 3D rotations. Fig. 5.55.6 shows an example adaptation trajectory that failed. To mitigate this issue, we run multiple adaptation attempts with different, random initializations and pick the one with the lowest final error. In practice, this only takes around 5 attempts and around 1 to 2 seconds.

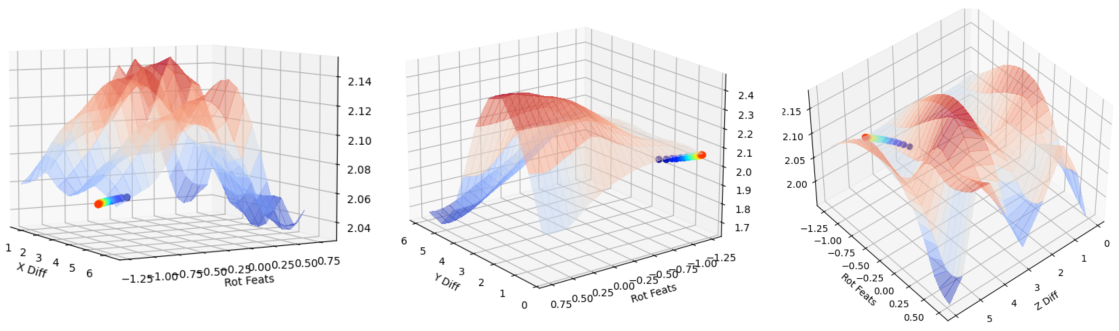

Figure 5.5: Loss surface plotted on rotation preference features versus absolute difference in rotational offsets as Euler angles X, Y, and Z respectively. An example convergence to poor local optima is shown by the scatter points with dark blue as the first and red as the final waypoint.

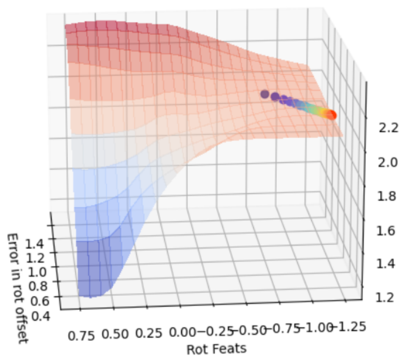

Figure 5.6: Loss surface defined on rotation preference feature versus angle from ground truth rotation offsets as quaternions. The same example optimization trajectory as Fig. 5.5 is shown.

5.4 Results

Now that we have explained the model implementation details used in our experiments, we discuss the results. After participants finished each task, they would fill out an anonymous survey to provide feedback with answers either descriptive or on a five-point scale.

Task 1:

1. Did they think the robot understood their intentions?
2. Did they exert a lot of effort to correct the robot?
3. Did the robot's reactions to their pushes match their expectation?
4. How satisfied were they about the robot's behavior?
5. Number of user interventions
6. Average distance of cup to table.

Figure 5.7 shows users felt our model understood their preferences better, required less effort, and were slightly more predictable after perturbations when compared to the baseline. Note that users were not told which system (baseline or our model) was being tested. Users generally felt that our model understood the "underlying objective of keeping the glass closer to the table". Two users noted that our model was sometimes unpredictable, lingering near the table too long and even ignoring the goal completely. This raises an important issue with over-fitting to short human perturbations and highlights our model's sensitivity to the hyper-parameters of adaptation learning rate and number of steps.

Task 2:

1. Did the robot present items how they wanted?
2. Error between robot and user's final perturbed orientation.

The 1st quartile, median, and 3rd quartile values for question one were 3.5, 4.0, and 4.0 respectively. For question two, we define orientation error between two quaternions q_1, q_2 as $\arccos(|q_1 \cdot q_2|) \in [0, \frac{\pi}{2}]$ [Huynh, 2009]. Average error was 0.2576, or 16% of the max possible error $\frac{\pi}{2} = 1.5708$. This indicates that our model matched users' desired orientations well.

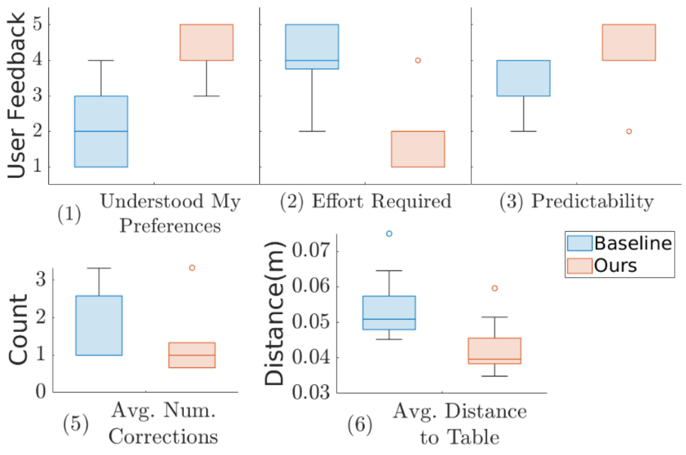

Figure 5.7: Task 1 user feedback for both baseline and our model, where each box plot is numbered according to the original list of Task 1 questions.

Task 3:

1. Did they think the robot understood their intentions?
2. How satisfied were they about the robot's behavior?

The 1st quartile, median, and 3rd quartile values for question one were 4.0, 4.0, and 5.0 respectively, indicating that users were very satisfied with our model's adaptability. This sentiment was also generally present in the second question. However, two users noted that the bowl should have moved closer to the scanner. This indicates that more gradent steps were needed, and again highlights the difficulty of tuning such parameters. A practical way to ensure the robot converges to an object is to simply treat it as a goal itself and define any complex task as a sequence of sub-goals.

5.5 Real Hardware Experiments

To truly evaluate our approach's effectiveness in interpreting *physical* human corrections, we also demonstrate OPA on a 7DOF Kinova robot arm in real life. In this scenario, a factory inspector must inspect and scan two categories of items: boxes and cans. A robot assists the human by picking up the items, presenting them with their barcode facing the human, and finally dropping them off in a bin. Boxes and cans have different barcode locations, however, so the human will need to correct the robot end effector orientation for each category. A collection of obstacles also lies on the robot's typical path, so the human must teach the preference of avoiding the obstacles. Starting with Fig. 5.8, the robot behaves incorrectly, presenting a box with the wrong orientation while also running into an obstacle. In Fig. 5.9, the human first teaches obstacle avoidance by pushing the robot away from the obstacle when it gets near. The human also teaches the robot to hold the box correctly in Fig. 5.10, and we overall see the adapted robot's behavior in Fig. 5.11. When all the boxes are finished scanning, the robot begins picking up cans. In our current formulation, the policy has no awareness of items picked up and only reasons about end effector motion relative to other objects. As a result, the human must demonstrate a new desired orientation in Fig. 5.12 for the can to be properly viewed. Fig. 5.13 shows the robot's adapted behavior, and Fig. 5.14 shows the adapted behavior but with a different human pose. The final scenario highlights the power of learning behavior relative to objects and their poses, which allows for zero-shot generalization to different object arrangements.

5.5.1 Implementation Details

In order to allow for safe human interaction with the robot, we implemented two modes of control on the robot: impedance control [Hogan, 1985] and gravity compensation. During human interventions, the robot uses only gravity compensation, and during robot control, the robot uses a combination of both, which ensures that the robot can correctly follow the policy's desired motion while still allowing for safe human contact. The human switches between these modes by pressing the spacebar on a keyboard. After human intervention ends, we run optimization for both position and orientation.

We encountered a scenario where the human may intend to correct only position or only orientation, but since the policy is not aware of this, both are adapted. For example, during adjustment of the box orientation, position would barely change, meaning that almost any position preference features would result in low loss. By naively trying multiple adaptation rounds and picking the solution with the lowest loss, the previously learned obstacle avoidance features would often be overwritten. To avoid this, we added a simple rule that requires loss to significantly decrease in order to be considered a valid solution.

The human sitting and standing poses were hard-coded for this demo, and we leave actual pose detection for future work. The box and can pick-up poses as well as obstacle position were also hard-coded, and we leave proper object detection for future work.

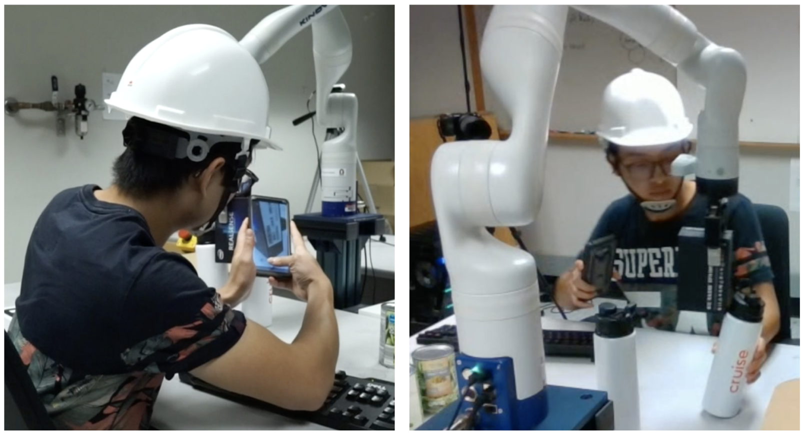

Figure 5.8: The robot presents a box in the wrong orientation, forcing the human to reach far to scan the box's label as shown on the left. The robot also almost knocks over one of the obstacles, a white bottle.

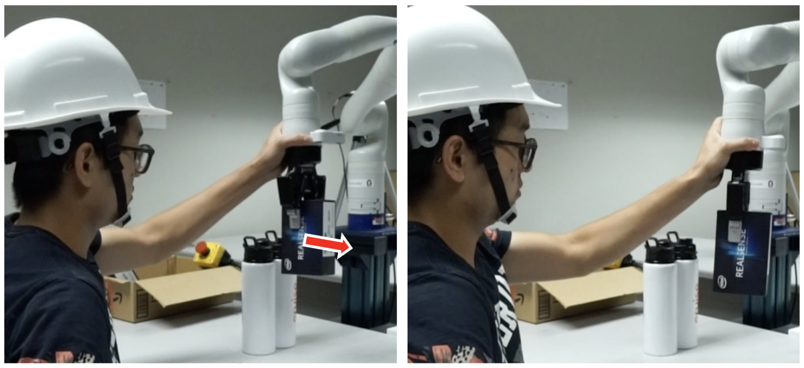

Figure 5.9: On the second box trial, the human first perturbs the robot to avoid colliding with the obstacle.

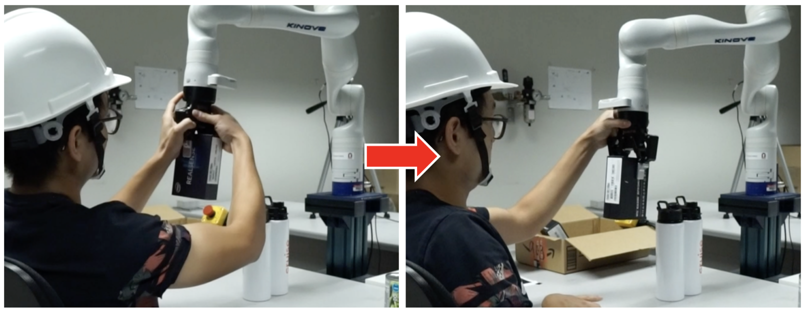

Figure 5.10: The human also perturbs the orientation of the box so that its barcode correctly faces the human.

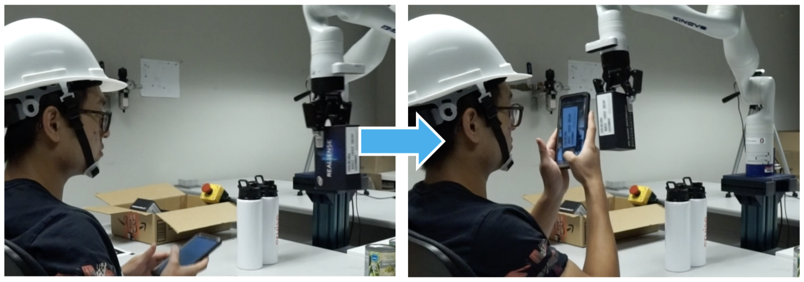

Figure 5.11: The robot shows adapted behavior on the third box, avoiding the obstacle and correctly presenting the box.

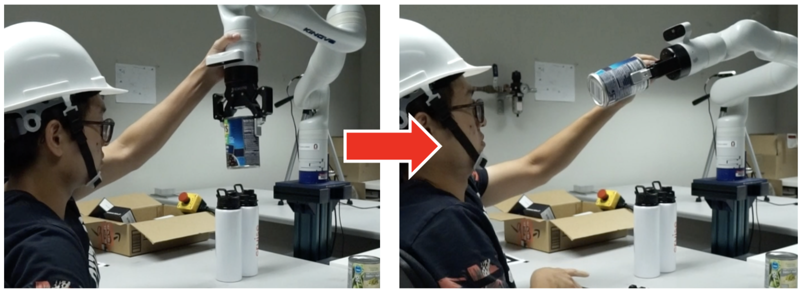

Figure 5.12: The robot presents a can in the wrong orientation, so the human perturbs to easily see its barcode.

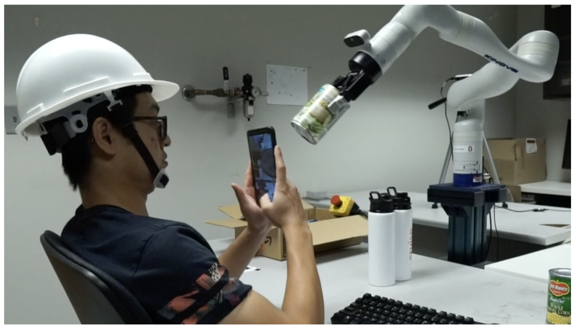

Figure 5.13: The robot shows adapted behavior for the next can.

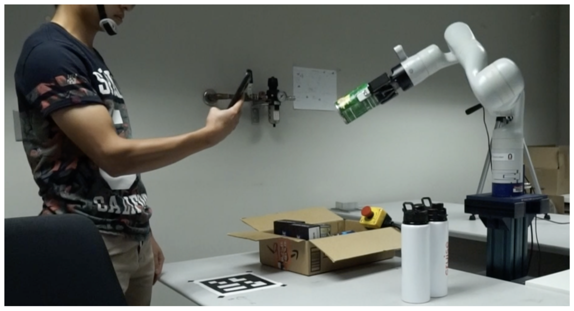

Figure 5.14: The robot presents a can to the human who is now standing with a different pose.

Chapter 6

Conclusions

We presented Object Preference Adaptation (OPA), a method for inferring human intentions for objects from physical corrections. OPA represents the environment as a graph with nodes as objects and edges between each object and the agent. Our method was able to fine-tune wrong behavior and learn new complex tasks in our experiments, all from a few physical perturbations. After pre-training an expressive base policy, OPA only needs to optimize object-specific features in a compact latent space, allowing for fast, effective adaptation. Robots and really any deep learning model must be flexible to ever-changing tasks and environments; we believe the key is to separate fixed, fundamental dynamics from task, environment, or object-specific behavior.

OPA relies heavily on object-centric actions and frames tasks as reach and avoid. Many tasks, however, also require fine-tuned grasping and even application of force (ie: wiping a surface). It would be exciting to extend this approach to also handle these complex behaviors. Tasks are not always object-relative but can be task-centric, such as drawing; the combination of both paradigms would be exciting.

Appendix A

Ablation Studies

A.1 Comparing Optimizers

In the previous experiments, we used Adam to adapt the learned preference features. This choice was motivated by Adam's popularity and strong performance in training deep neural networks. Other alternatives exist, however, including Recursive Least Squares (RLS) commonly used for adaptive filtering as well as the learned optimizer, Learn2Learn [Andrychowicz et al., 2016], which has been shown to outperform Adam on MNIST and CIFAR classification tasks. In this ablation study, we compare the performance of these two optimizers with Adam in terms of 1) average final loss and 2) robustness to noisy gradients as a result of the noisy physical corrections from humans. Our objective was to find any characteristics in the ground truth trajectories that could indicate which optimizer should be used, which could serve as a guide for others.

A.1.1 Mathematical Comparison

We first compare the formulation of Adam and RLS that can explain the experimental results observed later on. Following [Reddi et al., 2018], we see that many optimizers follow the same general algorithm of calculating some "averaging" over past gradients. Let $x_t \in \mathbb{R}^d$ denote the parameters to optimize at step t and $g_t \in \mathbb{R}^d$ as its gradient with respect to some loss function $f(x_t)$. Let function ϕ_t compute an aggregate m_t

Algorithm 1 Generic Adaptive Optimizer

```
Input: $x_1$
for $t = 1$ to $T$ do
    $g_t = \nabla f(x_t)$
    $m_t = \phi_t(g_1, \cdots g_t)$ and $V_t = \psi_t(g_1, \cdots g_t)$
    $x_{t+1} = x_t - \alpha_t m_t / \sqrt{V_t}$
end for
```

over the past gradients $g_1, \ldots g_t$ and function ψ_t compute a multiplier V_t to scale m_t. With this information, optimizers typically use the update algorithm Alg. 1.

Specifically, Adam uses the following ϕ_t and ψ_t:

$$\phi_t(g_1, \ldots g_t) = (1 - \beta_1) \sum_{i=1}^{t} \beta_1^{t-i} g_{t,i} \tag{A.1}$$

$$\psi_t(g_1, \ldots, g_t) = (1 - \beta_2)\mathrm{diag}(\sum_{i=1}^{t} \beta_2^{t-i} g_i^2) \tag{A.2}$$

which are both exponential averages that can be expressed recursively as:

$$m_{t,i} = \beta_1 m_{t-1,i} + (1 - \beta_1) g_{t,i} \tag{A.3}$$

$$V_{t,i} = \beta_2 V_{t-1,i} + (1 - \beta_2) g_{t,i}^2 \tag{A.4}$$

RLS, however, uses the following updates:

$$\phi_t(g_1, \ldots g_t) = g_t \tag{A.5}$$

$$\psi_t(g1, \ldots, g_t) = diag(\sum_{i=1}^{t} \lambda^{t-i} g_i^2) \tag{A.6}$$

which evaluates recursively to

$$m_{t,i} = g_{t,i} \tag{A.7}$$

$$V_{t,i} = \lambda V_{t-1,i} + g_{t,i}^2 \tag{A.8}$$

This shows that besides the minor difference in how the weight β_2 and forgetting factor λ are used to compute V_t, the key difference is that Adam uses an exponential

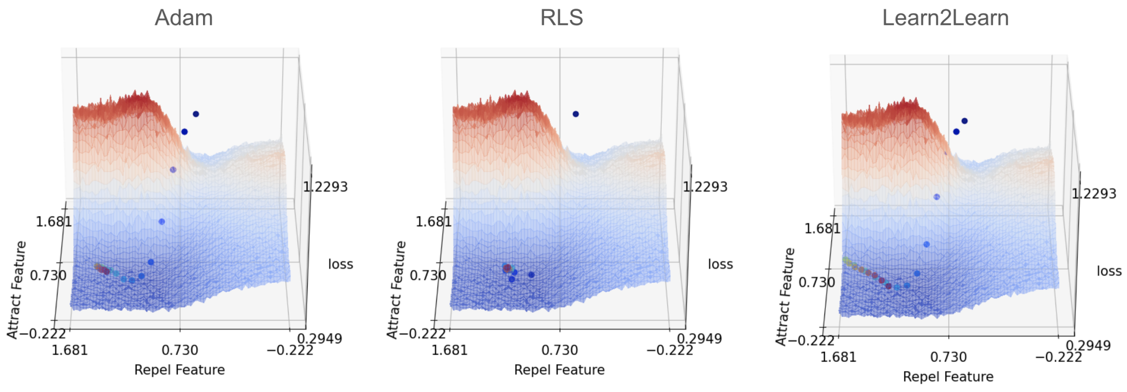

Figure A.1: The three plots show different optimization trajectories from Adam, RLS, and Learn2Learn for a random test scenario, where dark blue points are earliest, and red is the final optimized loss. A loss surface averaged over all position test samples is also plotted. The test scenarios used contain two objects that always correspond to attract and repel. This is why the loss surface is defined over an attract and a repel feature, and loss is minimized when the estimated feature of the attract object matches the actual attract feature value of -0.222 and vice versa for the repel object.

weighted average of gradients m_t, also known as momentum, whereas RLS only uses the immediate step's gradient g_t. The effects of momentum are well-studied [Qian, 1999]:

1. Average out oscillations along the directions perpendicular to the direction of steepest descent
2. Adding up contributions along the direction of steepest descent
3. Overall faster convergence to optimum

A.1.2 Experimental Comparison

We now explain how we experimentally compared all three optimizers and determine whether the results align with our observations from the mathematical comparison. To evaluate an optimizer on a given sample, we initialized the object preference features c_i and iteratively updated these features for a fixed number of steps using gradients with respect to the losses for position Eq. 4.4 and rotation Eq. 4.5. For parameter initialization, we set the initial position preferences as a fixed, weighted average between trained attract and repel features. Rotation preferences were similarly initialized between final care and ignore features. Rotational offsets were initialized as

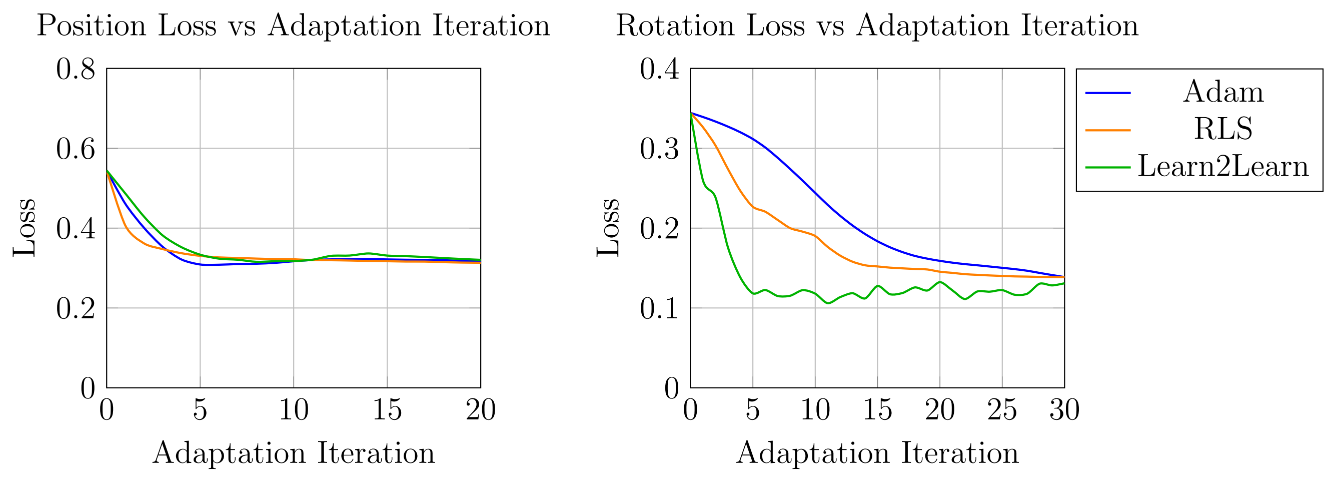

Figure A.2: Comparing average loss versus adaptation step over original test data.

zero rotation. Position features were adapted for 20 steps and rotation for 30 steps.

To fairly compare the three optimizers, we manually tuned each optimizer's hyperparameters on the training dataset. We then kept these fixed for final evaluation on the test set. For RLS, this includes forgetting factor $\lambda = 0.9$ and learning rate $\alpha = 0.5$. For the LSTM optimizer, this involved training the LSTM to minimize the adaptation loss for a fixed number of adaptation steps. For Adam, we used the default parameters as in previous experiments.

To finally evaluate each optimizer's performance, we measured loss versus adaptation step averaged over the entire test dataset, as shown in Fig. **??**. From the figure, we can see that Adam and Learn2Learn have generally more oscillations in loss than RLS throughout the optimization. However, it is possible that this is only a consequence of the chosen hyperparameters. Overall, all three optimizers have similar final performance.

Besides comparing average performance, we also compared the outputs of each optimizer with a batch gradient at each timestep to visualize the behavior of the optimizers. Comparing figures A.3, A.4, and A.5, we can clearly see the oscillation in adapted preference features where the learned values overshoot some local optima before slowly returning. We can see that RLS shows no large oscillations, but directly and quickly converges to the local optima. We also observe that Learn2Learn also

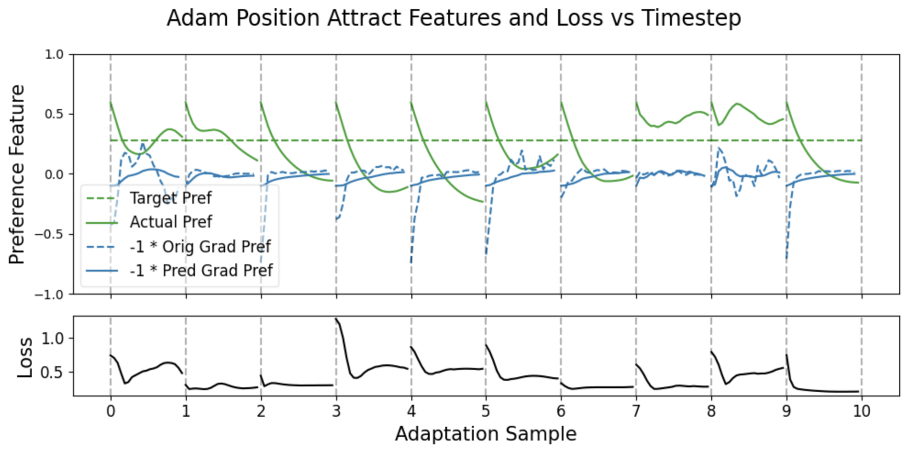

Figure A.3: Plot of Adam's performance on sample position data with the need to learn an attraction object. The top and bottom plots have aligned x-axes of adaptation or time step. The top plot compares actual learned preference in solid green with an approximate target value as well as original gradient versus predicted update. The bottom plot shows loss versus adaptation step. The dashed vertical lines separate different adaptation trials.

learns to apply some form of momentum term. A.1

These samples contain no noise, however, and rather only smooth expert trajectories generated from Gaussian Process interpolation. Since one key benefit of momentum is the ability to cancel out noise from directions perpendicular to the steepest descent, we examined each optimizer's performance when random Gaussian noise was added to each waypoint in the expert trajectories. This is especially important since human corrections often are not be perfect due to difficulty handling the robot or just unclear corrections. For each waypoint x_k^*, we added random noise $\epsilon_k \sim \mathcal{N}(0, \sigma I_3)$ to position with an example of $\sigma = 0.1$ shown in Fig. A.6.

Looking at Fig. A.7, we see that RLS is not robust to noise with standard deviation $\sigma = 0.1$ and fails to decrease loss, whereas Adam and Learn2Learn still on average decrease loss. Comparing actual gradients versus optimizer outputs in figures A.10, A.11, and A.12, we can see that RLS's outputs oscillate significantly whereas Adam's and Learn2Learn's are more smooth.

Beyond simply adding artificial noise, we wanted to see if any natural properties of the expert trajectories could cause better or worse performance for each optimizer. We first wanted a measure of "complexity" in the expert trajectories, where a straight

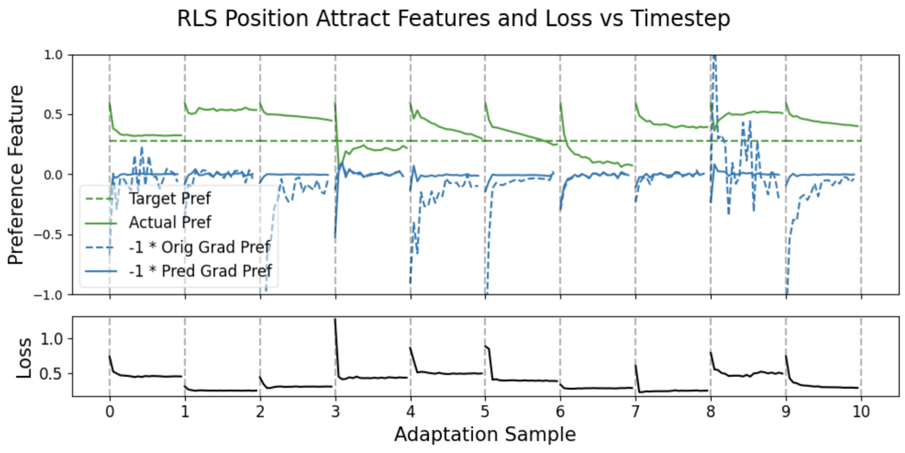

Figure A.4: Plot similar to A.3 but with RLS as the optimizer.

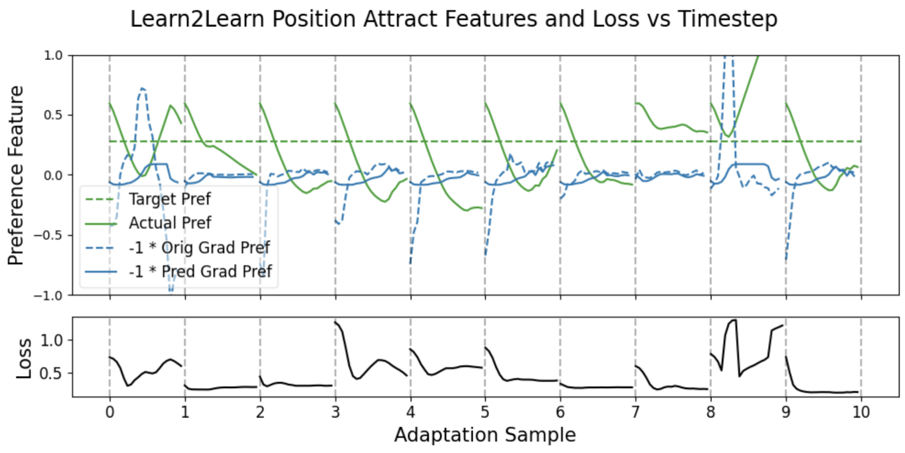

Figure A.5: Plot similar to A.3 but with Learn2Learn as the optimizer.

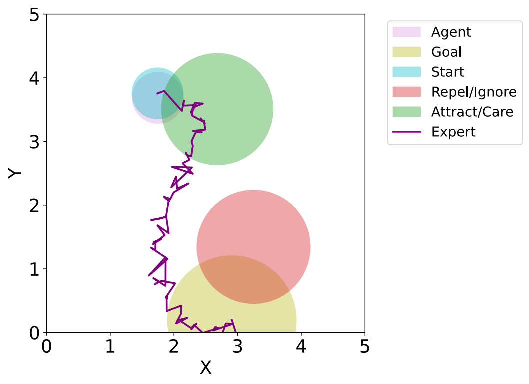

Figure A.6: Example expert trajectory shown in purple with random noise added.

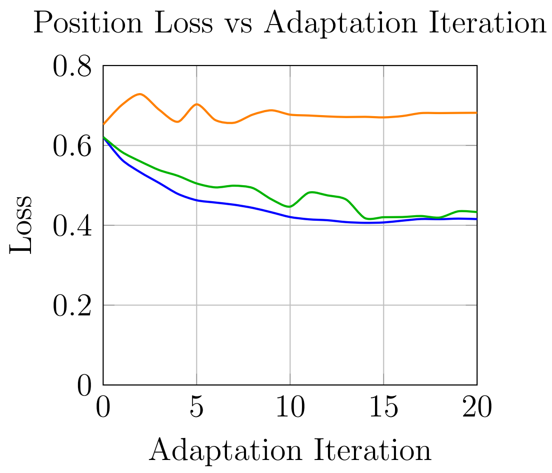

Figure A.7: Comparing average position loss versus update iteration with noise of $\sigma = 0.1$ added to ground truth trajectories.

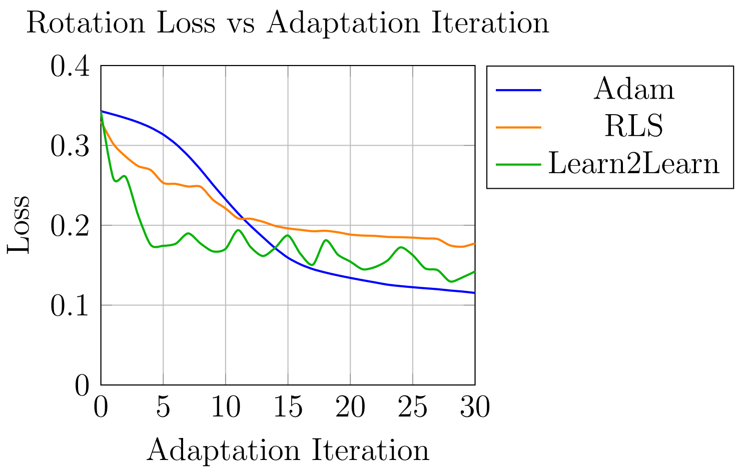

Figure A.8: Comparing average rotation loss versus update iteration with noise of $\sigma = 15$ degrees added to ground truth trajectories.

Figure A.9: Comparing average loss versus adaptation step over original test data.

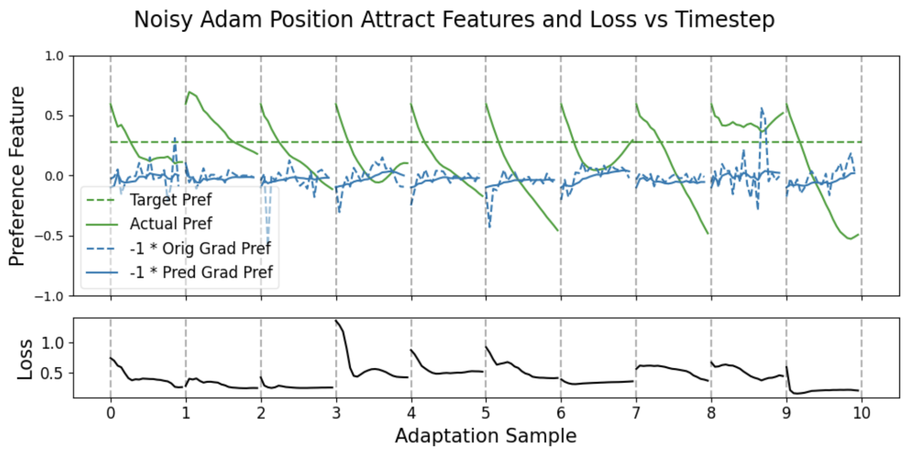

Figure A.10: Plot similar to A.3 but with added Gaussian noise.

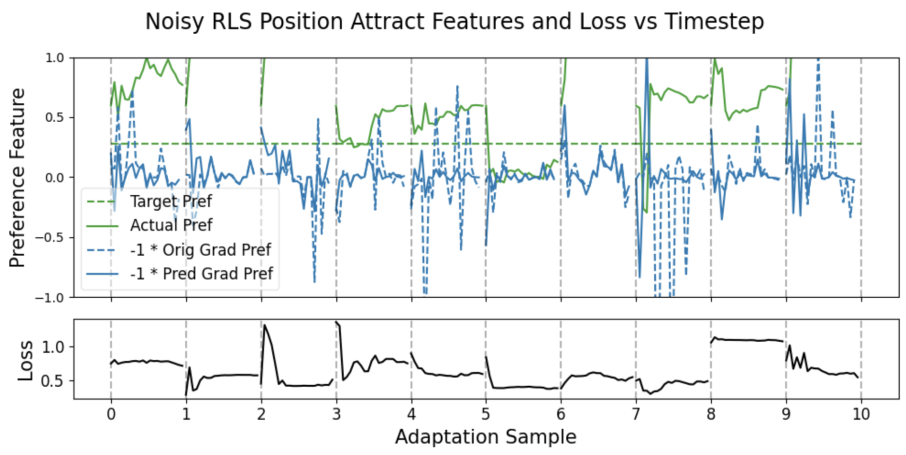

Figure A.11: Plot similar to A.3 but with RLS as the optimizer and added Gaussian noise.

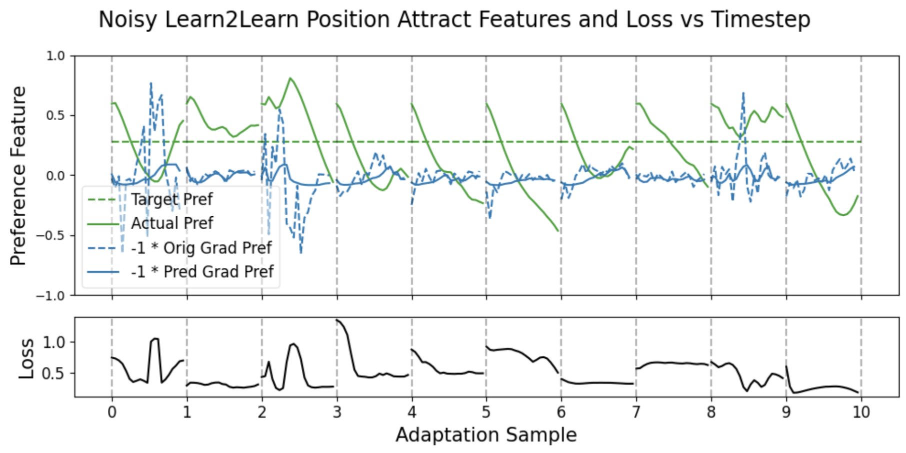

Figure A.12: Plot similar to A.3 but with Learn2Learn as the optimizer and added Gaussian noise.

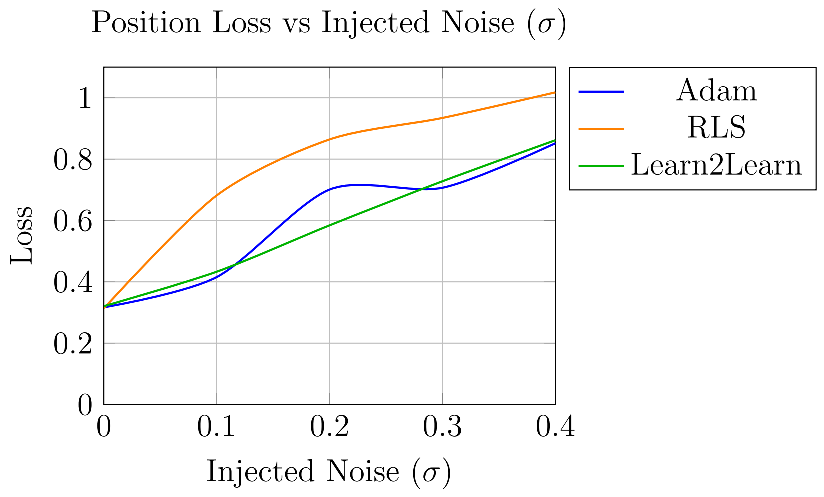

Figure A.13: Comparing average position loss versus standard deviation of artificial trajectory noise.

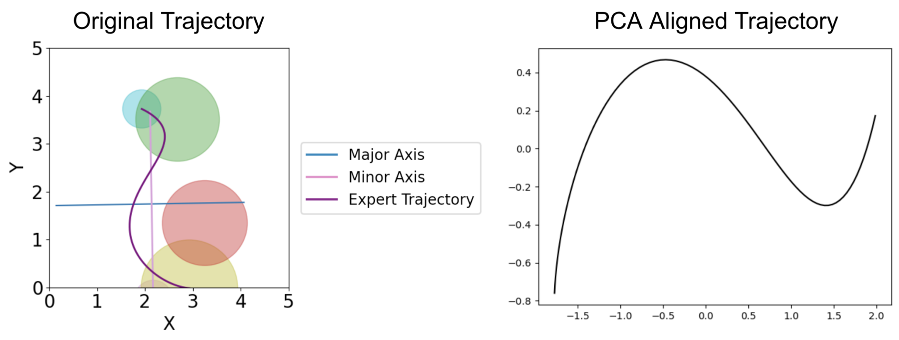

Figure A.14: Comparing average loss versus variance of the trajectories.

line would be the simplest. For each trajectory, we computed its major PCA axis and aligned the trajectory with respect to this axis, as shown in Fig. A.14. Given all aligned trajectories, we then computed variance along the y axis to represent complexity.

In Fig. A.15, we compare the final loss of the three optimizers averaged across trajectories separated by their y-axis variance. These variance bins were found using Kmeans clustering. Performance was nearly the same for low variance trajectories, but RLS performed significantly better than Learn2Learn and slightly better than Adam on high variance samples. It makes sense that Learn2Learn learns to handle the average trajectory with lower variance and performs worse on the rarer, high variance samples.

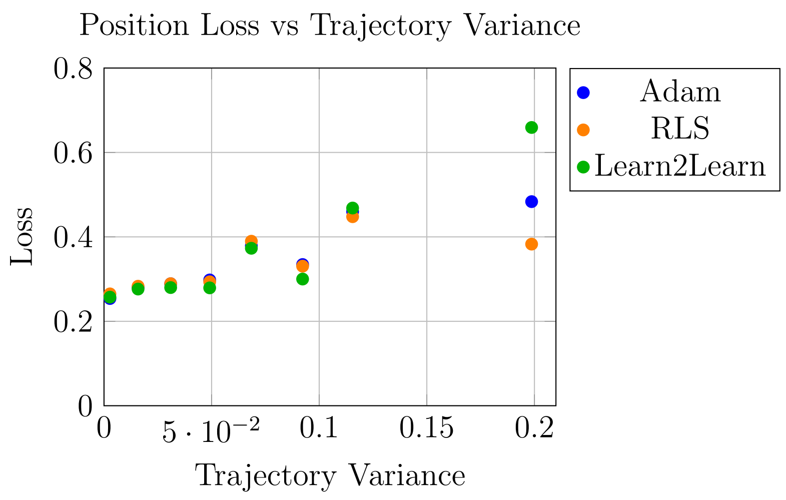

Figure A.15: Comparing average loss versus variance of the trajectories.

Optimizer	Run-time (sec)
Adam	0.005
RLS	0.243 (0.005)
Learn2Learn	0.005

Table A.1: Average run-time of Adam, RLS, and Learn2Learn. RLS has two run-times listed with the entry in parenthesis excluding Pytorch-specific implementation details.

Apart from adaptation performance, another key aspect in training deep neural networks as well as online adaptation is run-time. We calculated the run-time of one adaptation step for the three optimizers. The results, averaged across 10 samples of adaptation, are shown in table Table A.1.

We can see that RLS runs significantly slower than the Adam and Learn2Learn. However, this is likely caused by our specific implementation, which relies on Pytorch's auto-differentiation to calculate gradients. Pytorch easily calculates gradients of parameters with respect to a loss $\frac{\partial L}{\partial \theta}$, but not the actual outputs $\frac{\partial \hat{y}}{\partial \theta}$. As a result, the run-time in parenthesis only includes gradient computation with respect to one output $\hat{y}_j$ rather than all $\hat{y}$. This is because much of the intermediate gradients $\frac{\partial z_{N-1}}{\partial \theta_i}$

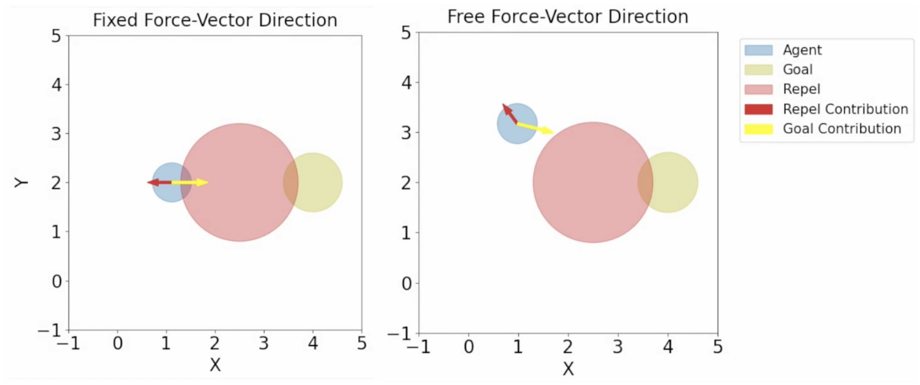

Figure A.16: Comparing behavior of a Potential Field (left) with OPA (right)

are shared:

$$\frac{\partial z_{N-1}}{\partial \theta_i} = \frac{\partial z_{N-1}}{\partial z_{N-2}} \frac{\partial z_{N-2}}{\partial z_{N-3}} \cdots \frac{\partial z_2}{\partial z_1} \frac{\partial z_1}{\partial \theta_i} \tag{A.9}$$

$$\frac{\partial L}{\partial \theta_i} = \sum_j \frac{\partial L}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_{N-1}} \frac{\partial z_{N-1}}{\partial \theta_i} \tag{A.10}$$

Pytorch typically computes $\frac{\partial z_{N-1}}{\partial \theta_i}$ only once, but for Jacobian calculation with respect to outputs, we had to manually recalculate these entries for every output $\hat{y}_j$. As a result, RLS run-time in parenthesis provides a more fair comparison, which is reasonable given that Adam performs very similar computations.

A.2 “Singularities” of Potential Fields

In Section 3-B of the paper, we mentioned how the position relation network overall outputs a push-pull force on the agent. We allow this direction to be freely determined by the network. Potential field methods also use this approach, but constrain the direction to be parallel to each agent-object vector. Only magnitude and sign of the vector can change. This may seem like an intuitive way to enforce structure in the network and reduce complexity, but this constraint fails during “singularities” where no orthogonal component is available to avoid an obstacle lying in the same direction as the goal.

Fig. A.16 compares ”forced” and ”free” direction behavior respectively. As the blue agent moves to the yellow goal in both cases, pay attention to the ”force” contribution

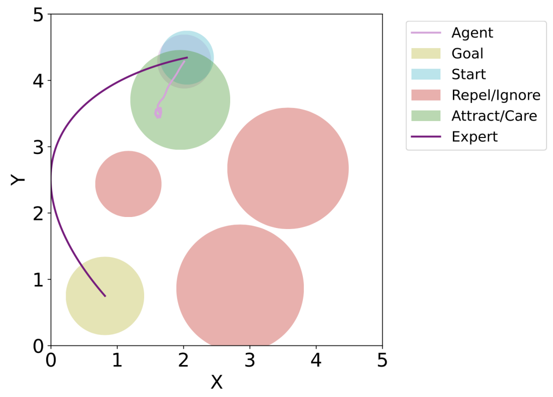

Figure A.17: The model's rolled out trajectory, shown in pink, converges to a "local optima" due to the all object influences canceling each other out.

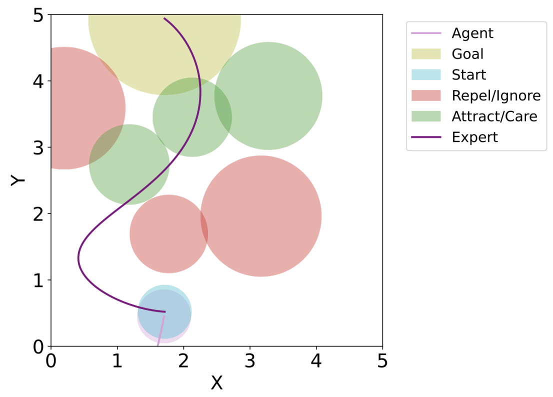

Figure A.18: With several close obstacles, the model prioritizes only the obstacles and does not converge to the goal.

of the goal and the repel object shown as yellow and red arrows respectively. In the "forced" version, the model correctly predicts a force vector pointing away from the red repel object. However, since there is no orthogonal component, the blue agent cannot avoid moving straight through the repel object as the goal force vector dominates. On the right side, however, the force direction contributed by the repulsor object has an orthogonal component, allowing the agent to avoid the repulsor.

A.3 Scalability with Number of Objects

Graph-based models and neural networks have an advantage of being computationally invariant to the order and number of objects in a scene. However, we only trained our model in scenarios with one or two objects, so it is unclear how the model will perform with more objects. We examined several scenarios with more objects and found that the model occasionally fails to converge to the goal. In some scenarios such as Fig. A.17, the force influences of surrounding objects overall cancel each other out, causing the agent to get stuck in one region. In other scenarios, the model's attention mechanism behaves strangely, where far away objects have unreasonably strong influence on the agent's motion, such as Fig. A.18.

We have several ideas on how this could be addressed:

1. Fine-tune the entire network, not just the object features, on new scenarios with increasing number of objects in a curriculum style. Since the goal is currently indistinguishable from other objects besides its feature value, this curriculum learning could change the goal feature to have more dominant influence on the agent. This does not completely eliminate convergence issues, but can mitigate them.

2. During training, we use simple Behavioral Cloning (BC). Networks trained with this approach are known to struggle in regions of the state space not covered by expert demonstrations since the policy is only evaluated along expert trajectories Ross et al. [2010]. One way to correct this issue is to use DAGGER Ross et al. [2010] in addition to BC during our pre-training phase. DAGGER is only feasible because our expert trajectories are synthetically generated in reasonable time, so there is no burden in querying for expert actions along the policy's rollout states.

3. Our treatment of the goal as a node like any other object is not ideal since interaction with the goal is distinct: we want to enforce a hard constraint of eventually converging to the goal. Perhaps an approach would be to remove the goal as a node, and instead have separate module to ensure convergence to the goal. We could follow a similar approach to how Dynamic Movement Primitives use a pseudo time scale $x \in [0, 1]$ that forces convergence to the goal.

Bibliography

Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. ICML '04, page 1, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138385. doi: 10.1145/1015330.1015430. URL https://doi.org/10.1145/1015330.1015430. 2, 2.1, 2.2

Aniket Agarwal, Ayush Mangal, and Vipul. Visual relationship detection using scene graphs: A survey, 2020. 2.4

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent, 2016. A.1

Andrea Bajcsy, Dylan P. Losey, Marcia K. O'Malley, and Anca D. Dragan. Learning robot objectives from physical human interaction. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 217–226. PMLR, 13–15 Nov 2017. 1, 2.1, 2.2

Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics, 2016. 2.4

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, 2018. 2.4

Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and Daniel Silver, editors, *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 17–36, Bellevue, Washington, USA, 02 Jul 2012. PMLR. URL https://proceedings.

mlr.press/v27/bengio12a.html. 1

Andreea Bobu, Andrea Bajcsy, Jaime F. Fisac, and Anca D. Dragan. Learning under misspecified objective spaces, 2018. URL https://arxiv.org/abs/1810.05157. 2.2

Andreea Bobu, Marius Wiggert, Claire Tomlin, and Anca D. Dragan. Feature expansive reward learning. *Proceedings of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*, Mar 2021. doi: 10.1145/3434073.3444667. 1, 2.3

Andreea Bobu, Marius Wiggert, Claire Tomlin, and Anca D. Dragan. Inducing structure in reward learning by learning features. *The International Journal of Robotics Research*, April 2022. doi: 10.1145/3434073.3444667. 1, 2.3

Daniel S. Brown, Yuchen Cui, and Scott Niekum. Risk-aware active inverse reinforcement learning, 2019a. URL https://arxiv.org/abs/1901.02161. 2.1

Daniel S. Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations, 2019b. URL https://arxiv.org/abs/1904.06387. 2.1

Daniel S. Brown, Russell Coleman, Ravi Srinivasan, and Scott Niekum. Safe imitation learning via fast bayesian reward inference from preferences, 2020. URL https://arxiv.org/abs/2002.09089. 2.3

Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2017. URL https://arxiv.org/abs/1706.03741. 2.1

Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization, 2016. URL https://arxiv.org/abs/1603.00448. 1, 2.3

Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning, 2017. URL https://arxiv.org/abs/1710.11248. 2.3

Rick Groenendijk, Sezer Karaoglu, Theo Gevers, and Thomas Mensink. Multi-loss weighting with coefficient of variations, 2020. 4.7

Luis Haug, Sebastian Tschiatschek, and Adish Singla. Teaching inverse reinforcement learners via features and demonstrations, 2018. URL https://arxiv.org/abs/1810.08926. 2.2

Neville Hogan. Impedance Control: An Approach to Manipulation: Part II—Implementation. *Journal of Dynamic Systems, Measurement, and Control*, 107(1):8–16, 03 1985. ISSN 0022-0434. doi: 10.1115/1.3140713. URL https://doi.org/10.1115/1.3140713. 5.5.1

Du Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35:155–164, 10 2009. doi: 10.1007/s10851-009-0161-2. 5.4

Ashesh Jain, Shikhar Sharma, Thorsten Joachims, and Ashutosh Saxena. Learning preferences for manipulation tasks from online coactive feedback, 2016. 2.1, 2.2

Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots, 2021. 2.5

Jean-Claude Latombe. *Potential Field Methods*, pages 295–355. Springer US, Boston, MA, 1991. ISBN 978-1-4615-4022-9. doi: 10.1007/978-1-4615-4022-9_7. 4.4

Manuel Lopes, Francisco Melo, and Luis Montesano. Active learning for reward estimation in inverse reinforcement learning. ECMLPKDD'09, page 31–46, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 3642041736. 2.1

Dylan P. Losey and Marcia K. O'Malley. Including uncertainty when learning from human corrections, 2018. 2.1

Dylan P. Losey and Marcia K. O'Malley. Learning the correct robot trajectory in real-time from physical human interactions. *J. Hum.-Robot Interact.*, 9(1), December 2019. doi: 10.1145/3354139. 1, 2.1, 2.2

Ajay Mandlekar, Jonathan Booher, Max Spero, Albert Tung, Anchit Gupta, Yuke Zhu, Animesh Garg, Silvio Savarese, and Li Fei-Fei. Scaling robot supervision to hundreds of hours with roboturk: Robotic manipulation dataset through human reasoning and dexterity. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1048–1055. IEEE, 2019. 4.7

Shaunak A. Mehta and Dylan P. Losey. Unified learning from demonstrations, corrections, and preferences during physical human-robot interaction, 2022. URL `https://arxiv.org/abs/2207.03395`. 1, 2.3

Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J. Andrew Bagnell, Pieter Abbeel, and Jan Peters. 2018. 2

Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(98)00116-6. URL `https://www.sciencedirect.com/science/article/pii/S0893608098001166`. A.1.1

S. Quinlan and O. Khatib. Elastic bands: connecting path planning and control. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 802–807 vol.2, 1993. doi: 10.1109/ROBOT.1993.291936. 4.7

Sashank Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018. A.1.1

Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2010. URL https://arxiv.org/abs/1011.0686. 2

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control, 2018. 2.4

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks, 2020. 2.4

Mohit Sharma, Jacky Liang, Jialiang Zhao, Alex LaGrassa, and Oliver Kroemer. Learning to compose hierarchical object-centric controllers for robotic manipulation, 2020. 2.4

Maximilian Sieb, Zhou Xian, Audrey Huang, Oliver Kroemer, and Katerina Fragkiadaki. Graph-structured visual imitation, 2020. 2.4

Jonathan Spencer, Sanjiban Choudhury, Matt Barnes, Matthew Schmittle, Mung Chiang, Peter Ramadge, and Siddhartha Srinivasa. Learning from interventions: Human-robot interaction as both explicit and implicit feedback. 07 2020. doi: 10.15607/RSS.2020.XVI.055. 4.8

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017. URL https://arxiv.org/abs/1703.06907. 1

Markus Wulfmeier, Dominic Zeng Wang, and Ingmar Posner. Watch this: Scalable cost-function learning for path planning in urban environments, 2016. URL https://arxiv.org/abs/1607.02329. 1, 2.3

Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification, 2017. 2.5

Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning, 2008. 2.2