Model-Based Inverse Reinforcement Learning from Visual Demonstrations

Neha Das[1,*] neha.das@tum.de
Sarah Bechtle[1,2,*] sbechtle@tuebingen.mpg.de
Todor Davchev[3] t.b.davchev@ed.ac.uk
Dinesh Jayaraman[1,4] dineshj@seas.upenn.edu
Akshara Rai[1] akshararai@fb.com
Franziska Meier[1] fmeier@fb.com

Abstract: Scaling model-based inverse reinforcement learning (IRL) to real robotic manipulation tasks with unknown dynamics remains an open problem. The key challenges lie in learning good dynamics models, developing algorithms that scale to high-dimensional state-spaces and being able to learn from both visual and proprioceptive demonstrations. In this work, we present a gradient-based inverse reinforcement learning framework that utilizes a pre-trained visual dynamics model to learn cost functions when given only visual human demonstrations. The learned cost functions are then used to reproduce the demonstrated behavior via visual model predictive control. We evaluate our framework on hardware on two basic object manipulation tasks.

Keywords: inverse RL, LfD, visual dynamics models, keypoint representations

1 Introduction

Learning from demonstrations is a very active area of research, motivated by enabling robots to bootstrap their learning processes. Demonstrations can help in various ways, for instance via inverse reinforcement learning (IRL), where the robot tries to infer the reward or goals from the human demonstrator. Most IRL approaches require demonstrations that couple action and state measurements, which are often costly to acquire.

In this work, we take a step towards model-based inverse reinforcement learning from visual demonstrations for simple object manipulation tasks. Model-based IRL approaches are thought to be more sample-efficient and hold promises for easier generalization [1]. Yet, thus far, their model-free counter-parts have been more successful in real world robotics applications with unknown dynamics [2, 3, 4]. In the context of learning cost functions from visual demonstrations, several major challenges remain for model-based IRL: most prior work [5, 6, 7] assume the transition model (of the environment and the robot) to be known, i.e. the robot knows how its actions change the environment state. However, when manipulating objects, the robot typically does not have access to such a model. Furthermore, IRL approaches typically involve a computationally intensive optimization procedure with an outer loop that estimates new cost function parameters, and an inner loop that solves the RL problem given the new cost function. The results in algorithms that do not scale well.

Our work targets both of these challenges, i.e. unknown transition models and computationally intensive optimization: 1) We present a system that enables visual inverse reinforcement learning from just a few human demonstrations. We utilize a vision module that extracts low-dimensional vision features both on human demonstrations, as well as on the robot. We pre-train a dynamics model with which the robot can predict how its actions change this low-dimensional feature representation. Once the robot has observed a latent state trajectory from a human demonstration, it can use its own dynamics model to optimize its actions to achieve the same (relative) latent-state trajectory. 2) We introduce a novel inverse reinforcement learning algorithm that enables learning cost functions

[1]Facebook AI Research, [2]MPI for Intelligent Systems, [3]University of Edinburgh, [4]University of Pennsylvania, *work done while at FAIR

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

from few demonstrations. Our IRL algorithm builds on recent progress in gradient-based bi-level optimization [8], which allows us to compute gradients of cost function parameters as a function of the inner loop policy optimization step, leading to more stable and effective optimization.

We evaluate our approach by collecting human demonstrations for two basic object manipulation tasks, learn the cost functions for these tasks and reproduce similar behaviors on a Kuka iiwa.

2 Background and Related Work

The proposed framework builds upon approaches from visual model-predictive control and IRL. This section provides an overview of the related methods and positions our work in context.

2.1 Visual Model Predictive Control

At the core of this paper lies the ability to optimize action sequences that minimize a task cost under a given visual dynamics model. Here we highlight how related work optimizes such action sequences and what cost function representations were chosen. [9, 10] optimize action sequences by utilizing the cross entropy method [11]. They learn pixel-level transition models and present methods for designing cost functions that evaluate progress to goal pixel positions, registration to goal images, and success classifiers. In contrast, [12] optimizes actions using gradient based methods, and learns a structured deep dynamics model that predicts change in the learned pose space, given applied actions. [13] learns locally linear dynamics models from images, and use stochastic optimal control algorithms in conjunction with quadratic cost functions (that penalize distances in latent space).

The above approaches either learn a dynamics model directly in pixel space or jointly learn a latent-space encoding and a dynamics model in that space. This is in contrast to recent work [14], which proposes a multi-step framework that consists of object segmentation and category level semantic 3-D keypoint detection, trained via supervision. In such a setup, actions are treated as transformations between start and goal pose of the 3-D keypoints while the cost function for optimizing the action is specified by the modeler in the form of geometric constraints governing the semantic 3-D keypoints. In this work, we combine ideas from all of these frameworks and extend them to the IRL domain. First, we train 2-D keypoint representations of images via self-supervised training [15, 16, 17]. Next, we train a dynamics model in that latent-space and optimize actions via gradient-based methods, similar to [12]. This differentiable action optimization is key to our IRL approach (Section 4.1).

2.2 Inverse Reinforcement Learning

Scaling inverse reinforcement learning to manipulation tasks in the physical world has proven difficult. This section provides an overview of some of the previously proposed methods, and positions our work in context. Model-free inverse reinforcement learning algorithms have been shown some success on real robotic platforms for manipulation tasks [2, 3, 4]. Kalakrishnan et al. [2] and Boularias et al. [3] only utilize proprioceptive state measurements and do not consider visual feature spaces.

However, most model-based IRL methods have been limited to simulation settings with known models [5, 18], and real robotics tasks with known models [6]. An exception is the work of Abbeel et al. [19] that learns dynamics models for helicopter flight tasks and then learns cost functions via apprenticeship learning [5]. Constrained optimization methods are a popular choice for IRL approaches [6, 20, 21]. Scaling such methods to image-based tasks is highly non-trivial. In contrast, we pre-train a visual dynamics model, and present a gradient-based IRL approach, which is built on recent successes in gradient-based bi-level optimization [8, 22].

IRL and Inverse Optimal Control (IOC) from Visual Demonstrations: There have been several approaches that utilize visual demonstrations to learn cost functions [7, 23, 24, 4]. [7, 23] learn cost functions for path planning tasks in urban and track environments, while Sermanet et al. [24] and Finn et al. [4] focus on manipulation tasks. [24, 4] employ a model-free IRL approach to learn reward functions from visual demonstrations. Both methods rely on kinesthetic demonstrations, either for the full IOC approach [4]; or to initialize the policy that optimizes the learned reward function [24]. In contrast, our approach is model-based. We only utilize expert demonstrations as part of the dynamics model training, and can extract cost functions from visual demonstrations only. When optimizing our policies we do not require expert data for initialization.

3 Gradient-Based Visual Model Predictive Control Framework

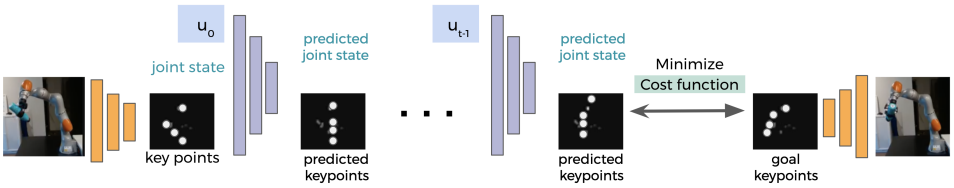


Figure 1: Overview of our keypoint-based visual model predictive control framework. Actions are optimized via gradient descent on the cost function.

In this section we describe our gradient-based visual model predictive control approach that combines recent advances in unsupervised keypoint representations and model-based planning. In the next section, we will build our novel inverse reinforcement learning system on top of this foundation.

The proposed system, depicted in Figure 1, comprises of following modules: 1) a keypoint detector that produces low-dimensional visual representations, in the form of keypoints, from RGB image inputs; 2) a dynamics model that takes in the current joint state $\theta, \dot{\theta}$ and actions u and predicts the keypoints and joint state at the next time step; and 3) a gradient based visual model-predictive planner that, given the dynamics model and a cost function, optimizes actions for a given task. Next, we provide a quick overview of each of these modules.

3.1 Keypoints as visual latent state and dynamics model

We use an autoencoder with a structural bottleneck to detect 2D keypoints that correspond to pixel positions or areas with maximum variability in the input data. The architecture of the keypoint detector closely follows the implementation in [15]. To train our keypoint detector we collect visual data $\mathscr{D}_{\text{key-train}}$ for self-supervised keypoint training (see Appendix A.2). After this training phase, we have a keypoint detector that predicts keypoints $z = g_{\text{key}}(o_{\text{im}})$ of dimensionality $K \times 3$. Here K is the number of keypoints, and each keypoint is given by $z_k = (z_k^x, z_k^y, z_k^\mu)$, where z_k^x, z_k^y are pixel locations of the $k-$th keypoint, and z_k^μ is its intensity, which corresponds roughly to the probability that that keypoint exists in the image.

Given a trained keypoint detector, we next collect dynamics data to train a dynamics model $\hat{s}_{t+1} = f_{\text{dyn}}(s_t, u_t)$. The dynamics model is trained to predict the next state, from current state s_t and action u_t, where the state $s_t = [z_t, \theta_t]$ combines the low-dimensional visual state $z_t = g_{\text{key}}(o_{\text{im},t})$ and the joint state θ_t. Actions u_t are desired joint angle displacements. For simple tasks we train this dynamics model on data generated through sine motions on the joints. However for complex tasks, we utilize expert demonstrations to learn this dynamics model.

3.2 Gradient-Based Visual MPC towards a keypoint goal state

We want to optimize an action sequence $\mathbf{u} = (u_0, u_1, \ldots, u_T)$ that moves the arm towards the visual goal keypoints z_{goal} extracted from a goal image. Similar to other visual MPC work [10, 12] we utilize our learned visual dynamics model f_{dyn} to optimize actions $\mathbf{u}$. Two ingredients are necessary to implement this step: 1) a cost function that measures distances in visual latent space; 2) an action optimizer that can minimize that cost function. We build on the gradient based action optimization presented in [12] and extend it for optimizing actions over a time horizon T. Specifically, to optimize a sequence of action parameters $\mathbf{u} = (u_0, u_1, \ldots, u_T)$ for a horizon of T time steps, we first predict the trajectory $\hat{\tau}$, that is created through the current $\mathbf{u}$ from starting configuration s_0: $\hat{s}_1 = f_{\text{dyn}}(s_0, u_0)$, $\hat{s}_2 = f_{\text{dyn}}(\hat{s}_1, u_1), \ldots$ $\hat{s}_T = f_{\text{dyn}}(\hat{s}_{t-1}, u_{t-1})$, which generates a predicted (or planned) trajectory $\hat{\tau}$. Intuitively, this step uses the learned dynamics model f_{dyn} to simulate forward what would happen if we applied action sequence $\mathbf{u}$. We then measure the cost achieved $C_\psi(\hat{\tau}, z_{\text{goal}})$ and perform gradient descent on actions $\mathbf{u}$ such that the cost of the planned trajectory is minimized

$$\mathbf{u}_{\text{new}} = \mathbf{u} - \eta \nabla_u C_\psi(\hat{\tau}, z_{\text{goal}}) \tag{1}$$

Details of our full visual MPC algorithm can be found in the Appendix, in Algorithm 3. Manually designing this cost function is hard, especially in visual feature spaces. In the next section we propose a gradient-based inverse reinforcement learning algorithm to learn this cost function.

4 Gradient-Based IRL from Visual Demonstrations

Most inverse RL algorithms have an inner and outer optimization loop; the inner loop optimizes actions or policies given the current cost function parameters ψ, and the outer loop optimizes the cost function parameters given the results of the inner loop. To the best of our knowledge, all existing IRL approaches implement these two optimization steps independently. As we show below, and in our experiments, this can lead to instability in the optimization. Here we derive an algorithm that optimizes cost parameters ψ as *a function* of the inner loop policy optimization step, such that updates to parameters ψ are directly related to their performance in the inner loop.

Algorithm 1 Gradient-Based IRL for 1 Demo

Initial ψ, pre-trained f_{dyn}, learning rates $\eta = .001, \alpha = .01$
demos $\tau_{\text{demo, i}}$, with goal state $z_{\text{goal}} = \tau_T$
initial state $s_0 = (\theta_0, \dot{\theta}_0, z_0)$
for each *epoch* **do**
  $u_t = 0, \forall t = 1, \ldots, T$
  for each i in $\text{iters}_{\text{max}}$ **do**
    // rollout $\hat{\tau}$ from initial state s_0 and actions u
    $\hat{\tau} \leftarrow \text{rollout}(s_0, u, f_{\text{dyn}})$
    // Gradient descent on u with current C_ψ
    $u_{new} \leftarrow u - \alpha . \nabla_u C_\psi(\hat{\tau}, z_{goal})$
  end for
  // Update ψ based on u_{new}'s performance
  $\hat{\tau} \leftarrow \text{rollout}(s_0, u_{\text{new}}, f_{\text{dyn}})$
  // Computes gradient through the inner loop
  $\psi \leftarrow \psi - \eta . \nabla_\psi \mathscr{L}_{IRL}(\hat{\tau}, \tau_{\text{demo}})$
end for

Specifically, in this work we address deterministic, fixed-horizon and discrete time control tasks with continuous states $\mathbf{s} = (s_1, \ldots, s_T)$ and continuous actions $\mathbf{u} = (u_1, ..., u_T)$. Each state $s_t = [\theta_t, \dot{\theta}_t, z_t]$ is the concatenation of the measured joint angles and velocities $\theta_t, \dot{\theta}_t$ and the extracted keypoints z_t at time step t. The control tasks are characterized by a pre-trained visual dynamics model $\hat{s}_{t+1} = f_{\text{dyn}}(s_t, u_t)$ and the learned cost function C_ψ.

4.1 Learning cost functions for action optimization

In our IRL algorithm, the outer loop optimizes cost parameters ψ and the inner loop optimizes actions $\mathbf{u}$ given the current cost. The result of the inner loop step is a predicted latent trajectory $\hat{\tau}$. Intuitively, we want to learn a cost function C_ψ, that, when used in the inner loop, minimizes the IRL loss $\mathscr{L}_{IRL}(\tau_{\text{demo}}, \hat{\tau})$ between $\hat{\tau}$ and the expert demonstrations τ_{demo}. To put it succinctly, we want to compute the gradient of $\mathscr{L}_{IRL}$ wrt to ψ: $\nabla_\psi \mathscr{L}_{IRL}$.

To compute this gradient, let's first consider a case where the demonstration consists of only one observation (e.g. the goal) $\tau_{\text{demo}} = s_{\text{demo}}$, and we want to optimize one action parameter u to achieve this goal in one time step. Then we can write out the IRL optimization problem as

$$\nabla_\psi \mathscr{L}_{IRL}(\tau_{\text{demo}}, \hat{\tau}_\psi) = \nabla_{\hat{\tau}_\psi} \mathscr{L}_{IRL}(\tau_{\text{demo}}, \hat{\tau}_\psi) \nabla_\psi \hat{\tau}_\psi \tag{2}$$

$$= \nabla_{\hat{\tau}_\psi} \mathscr{L}_{IRL}(\tau_{\text{demo}}, \hat{\tau}_\psi) \nabla_\psi f_{\text{dyn}}(s, u_{\text{opt}}) \tag{3}$$

$$= \nabla_{\hat{\tau}_\psi} \mathscr{L}_{IRL}(\tau_{\text{demo}}, \hat{\tau}_\psi) \nabla_\psi f_{\text{dyn}}(s, u_{\text{init}} - \eta \nabla_u C_\psi(s_{\text{demo}}, f_{\text{dyn}}(s, u)) \tag{4}$$

where in Eq 2 we apply the chain rule to decompose $\nabla_\psi \mathscr{L}_{IRL}(\tau_{\text{demo}}, \hat{\tau}_{C_\psi})$ into the gradient of $\mathscr{L}_{IRL}$ with respect to the predicted trajectory $\hat{\tau}_\psi$ and the gradient of $\hat{\tau}_\psi$ wrt cost parameters ψ. In the next step, Eq 3, we plug in the rollout of the predicted trajectory, which is only one time step, so $\hat{\tau}_\psi = f_{\text{dyn}}(s, u_{\text{opt}})$, where u_{opt} is the optimized action parameter. In the final step, Eq 4, we write out the gradient update of the action parameters u which shows the dependence on the cost function C_ψ.

This optimization problem is reminiscent of recent gradient-based bi-level optimization approaches to meta-learning [25, 22], involving two sets of parameters (in our case $\mathbf{u}, \psi$) to be optimized. Such gradient-based solutions typically involve tracking the gradients through the inner loop, and then auto-differentiating the inner loop optimization trace with respect to the outer parameters. We use the gradient-based optimiser *higher* [8] to tackle this bi-level optimization problem. We have described our gradient-based IRL algorithm for inner loops with one step optimization, the extension over multiple time steps requires Eq 3 and Eq 4 to be adapted to the predicted trajectory over T time steps. A high-level overview of our gradient-based IRL algorithm is presented in Algorithm 1.

4.2 Cost functions and IRL Loss for learning from visual demonstrations

Our algorithm depends on both, the specification of the IRL loss $\mathscr{L}_{IRL}$ and the cost function parametrization C_ψ. Intuitively, the $\mathscr{L}_{IRL}$ should measure the distance between the predicted latent trajectory $\hat{\tau}$ and the demonstrated latent trajectory τ_{demo}. We would like to keep the $\mathscr{L}_{IRL}$ as simple as possible, and thus choose it to be the squared distance between predicted and demonstrated keypoints at each time step, $\mathscr{L}_{IRL}(\tau_{\text{demo}}, \hat{\tau}) = \sum_t (z_{t,\text{demo}} - \hat{z}_t)^2$. Similar to [6], we compare three distinct parametrizations for the cost function C_ψ:

Weighted Cost $C_\psi(\hat{\tau}, z_{\text{goal}}) = \sum_k \left[\psi_k^x \sum_t (\hat{z}_{t,k}^x - z_{\text{goal},k}^x)^2 + \psi_k^y \sum_t (\hat{z}_{t,k}^y - z_{\text{goal},k}^y)^2 \right]$

where $\hat{z}_{t,k}^x, \hat{z}_{t,k}^y$ is the kth predicted keypoint at time-step t and $z_{\text{goal, k}}^x, z_{\text{goal, k}}^y$ is the goal keypoint. This simple cost function parametrization provides a constant weight per x, y dimension of each key point. This cost function has $K \times 2$ parameters.

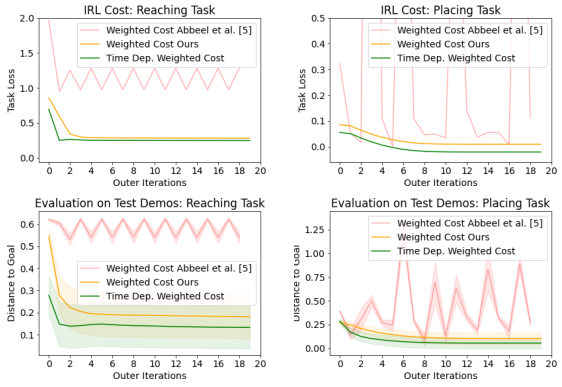
Time Dependent Weighted Cost $C_\psi(\hat{\tau}, z_{\text{goal}}) = \sum_k \sum_t \left[\psi_{t,k}^x (\hat{z}_{t,k}^x - z_{\text{goal},k}^x)^2 + \psi_{t,k}^y (\hat{z}_{t,k}^y - z_{\text{goal},k}^y)^2 \right]$

This cost extends the previous formulation to provide a weight for each time step t. This adds more flexibility to the cost and allows to capture time-dependent importance of specific keypoints. This cost function has $T \times K \times 2$ parameters, which scale linearly with the horizon length.

RBF Weighted Cost $C_\psi(\hat{\tau}, z_{\text{goal}}) = \sum_k \sum_t \sum_j \left[\psi_{j,k}^x(t) (\hat{z}_{t,k}^x - z_{\text{goal},k}^x)^2 + \psi_{j,k}^y(t) (\hat{z}_{t,k}^y - z_{\text{goal},k}^y)^2 \right]$

Here we introduce J time dependent RBF kernels $\psi_{j,k}(t) = \exp(b(t - \mu_j)^2)$. This cost allows us to more easily scale to longer time horizons, with $J \times K \times 2$ parameters, and $J < T$. Kernels are uniformly spaced in time and b is chosen to create some overlap between neighboring kernels.

4.3 Illustrative Comparison with Feature-Matching IRL



(a) (b)

Figure 2: (Top) IRL cost during cost training for reaching (a) and placing (b) task with one demonstration. (Bottom) Performance of learned cost on five test tasks. We compare our learned IRL costs with a cost trained using apprenticeship learning [5] .

Here, we illustrate the differences between our approach and feature matching IRL approaches in terms of optimization behavior on simulation tasks with known models. We compare our method to the IRL apprenticeship learning algorithm from [5] for a reaching and a placing task on a simulated Kuka robot. We adapted the code from [26] for our experiments (see appendix A.4). We assume to be given ground truth keypoints in 3-D, placed on an object that the Kuka holds. Furthermore we use a differentiable model that predicts keypoint changes for applied actions.

We train a weighted cost using only one demonstration, and evaluate the learned cost function on five test demonstrations for with our algorithm and our baseline. In Figure 2 we show convergence on training and test tasks, as a function of outer loop iterations. We see that our baseline oscillates between good and bad solutions, while our algorithm converges to a good solution. We believe this improvement in convergence behavior is due to the presence of an explicit connection between the policy optimization and the cost function parameter learning in our method. This connection allows us to compute gradients that communicate between inner and outer loops and thus explicitly account for the cost function performance for policy optimization during cost function learning. In contrast, existing model-based IRL approaches, such as the feature matching algorithm, separate the outer and inner loop and rely on careful design of multiple constraints or features to update cost function parameters.

5 Hardware Experiments

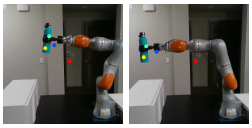


Figure 3: Reaching task

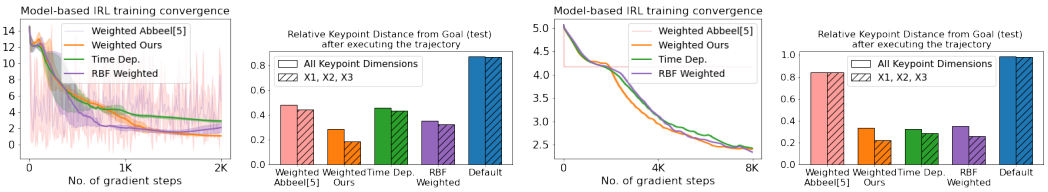
We evaluate the proposed approach for inverse reinforcement learning from visual demonstrations by performing a sequence of qualitative and quantitative experiments. We seek to interpret the learned cost functions and investigate their ability to successfully reproduce the demonstrated tasks. In our experiments, we assume we have pre-trained a key point detector (see Figure 3), and a good enough visual dynamics model to accomplish the task. We use the same keypoint detector for all experiments. Details about training the keypoint detector and the dynamics model can be found in the Appendix.

5.1 Quantitative Analysis on automatically generated visual demonstrations

We collect a set of 15 automatically generated demonstrations of moving an object from one (visual) location to another using the KUKA arm, making sure that visually the object moves only in the X-axis (see Figure 5 for an example). Constraining the movement of the gripped object in this way allows us to interpret the learned cost functions better. We also note that one of our 4 keypoints (in red), is fixed in the background. The collected demonstrations comprise the start state $\theta_0, \dot{\theta}_0$, and keypoint observations $z_t = g_{\text{key}}(o_{\text{im,t}})$, for $T = 25$ frames at a frame rate of $5Hz$. The keypoint detector predicts 4 keypoints per frame. We train the parametrized costs described in Section 4.2 with 1 and 10 reaching demonstrations; and evaluate their performance by optimizing an action policy using the learned costs on 5 test demonstrations.

We compare our IRL algorithm to 2 baselines: (1) the IRL apprenticeship learning algorithm [5] combined with the weighted cost from 4.2, and (2) a naive ("Default") cost that measures the distance between the predicted and goal keypoint. This cost is defined as $C_{\text{default}} = \sum_t^T (\hat{z}_t - z_{\text{goal}})^2$ for a trajectory with T steps. For visual model-predictive control via learned (or default) cost, the learning rate for action optimization is chosen to be the same as during the IRL training phase, $\eta = 0.001$.

5.1.1 Training and Analysis of the Cost Functions

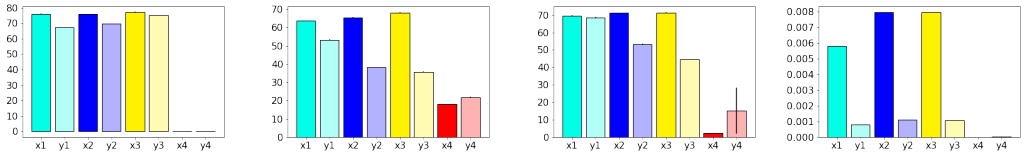


(a) 1 demo, train (b) 1 demo, test (c) 10 demos, train (d) 10 demos, test

Figure 4: **IRL training and test evaluation** (a) and (c) show the $\mathscr{L}_{IRL}$ during training of the parametrized costs from 1 and 10 demos. Figures (b, d) show the relative distance to the goal keypoint achieved at test time when optimizing the action trajectory with the learned costs and baselines. Results are averaged across 3 seeds.

Figure 4 depicts the results achieved on the simple reaching task. The final relative distance (see Appendix) to goal keypoint positions from the planned trajectory is considerably less when optimized using all three of the learned costs compared to both baselines (see (b, d)). We calculate and compare this metric for all keypoint dimensions as well as only the dimensions corresponding to $z^x_{t,1}, z^x_{t,2}$ and $z^x_{t,3}$, which are the least noisy keypoint observation dimensions. We also note that the learned costs perform overall similarly irrespective of whether they were trained on a single demonstration or on ten demonstrations. This observation encouraged the use of a single demonstration for the next set of experiments (Section 5.2), where such demonstrations are harder to acquire.

As noted before, the 10 reaching demonstrations we used for training had very little variability for the visual keypoints along the Y-direction and for one particular keypoint (marked red in Figure 3). Figure 5 (a,b,c) illustrate that all of the proposed parametrized cost functions learn relatively small weights corresponding to the Y-axes and the red keypoint. indicating that they have identified properties of the visual demonstrations they have been trained on. Finally, the parameters of baseline(1) (while being significantly smaller) have a similar weight structure to the rest of the models. This indicates that they are able to capture the demonstration properties. However, their overall



(a) Weighted (ours) (b) TimeDep (ours) (c) RBF Weighted (ours) (d) Weighted (baseline)

Figure 5: **Learned cost Parameters:** corresponding to the keypoint vector's dimensions after training on 10 demos. Y-axes and one keypoint (in red) receive less weight. Colors are matched to keypoints shown in Fig 3.

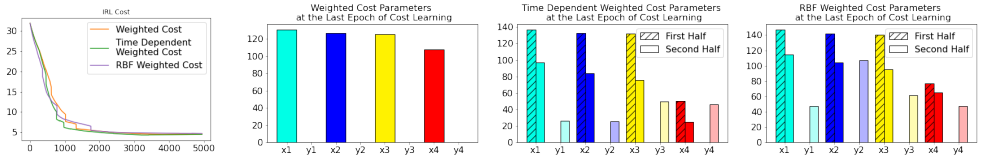
performance during evaluation was far worse than our learned costs due to the lesser weight each parameter bears. Note that we could tune the learning rate η with which actions are optimized at test time to account for these smaller weights, which would improve performance of our baseline. However, this is not necessary for our algorithm, which learned to scale cost function parameters wrt to the η used during the IRL training phase. Furthermore, the IRL optimization procedure for baseline (1) was very unstable, and it was unclear whether the algorithm has or will converge. Previous work has proposed to scale and regularise the learned weights as done in the maximum entropy literature [2, 3] or define additional constraints [6] to address some of these issues. We believe one reason for this instability is that the inner and outer loops are disconnected in such feature matching algorithms. Our algorithm instead connects the inner and outer loop optimization steps, and is therefore able to leverage gradient updates from action optimization in the inner loop for learning cost function parameters that automatically work well on the desired task without any additional help.

5.2 Learning Cost Functions from Visual Human Demonstrations

In this subsection we scale the proposed method to a more challenging task both from manipulation and demonstration points of view. We consider the task of placing a bottle on a shelf demonstrated by a human user through video data.

Expert Demonstration Data Collection We collect the human demonstration at a frame rate of 30 Hz, which we then downsample to 5 Hz. In contrast to Section 5.1, we do not have access to the initial proprioceptive state $\theta, \dot{\theta}$. We therefore test with 2 starting configurations of the robot. Start configuration 1) we choose an initial position for the robot that is roughly close to the human demonstration's initial position; and start configuration 2) that is closer to the target. We preprocess all the video-frames to obtain keypoint vectors corresponding to each step, relative to the first frame.

Training the Cost Functions and analysis We experiment on a task that is comprised of two individual motions. During the first half of the demonstration, the object moves only along the X-axis towards the shelf, while in the second half it moves downwards (i.e. along the Y-axis, while X-coordinate of the object remains constant). We train the 3 cost function architectures from Section 4.2 on a single human demonstration for placing a bottle for 5000 gradient steps.

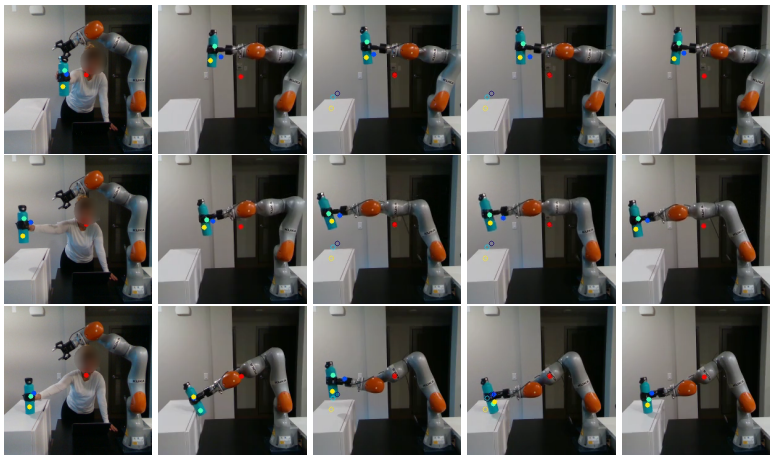


(a) Training: IRL Loss (b) Weighted (c) TimeDep Weighted (d) RBF Weighted

Figure 6: (a) plots the $\mathscr{L}_{IRL}$ while training costs for 5K gradient steps with a human demonstration. (b, c and d) show the values of the learned costs' parameters. For Time Dependent (b) and RBF costs with 5 kernels (c) which calculate separate parameters corresponding to each step of the trajectory, we compare the mean of the parameters corresponding to each keypoint across the first five steps to the last five steps.

The $\mathscr{L}_{IRL}$ loss converges roughly around $2K$ iterations (Figure 6 (a)). We note that the parameters of the time-dependent cost functions (Figure 6 (b and c)) learn to emphasize the distance from the goal in the X direction during the first half of the motion and Y-direction in the latter half.

5.2.1 Using the Learned Cost Function on the Robot



(a) Demo (b) Default (c) Weighted (d) TimeDep (e) RBF

Figure 7: Column a) Human Demonstration that is used for the IRL algorithm to extract cost functions. Column b)-d) Comparison of visual MPC result using the default and learned costs. First row corresponds to timestep $t = 0$, middle row to $t = 5$ and bottom row to $t = 10$ of executing the placing task. The detected and goal keypoints in each image are depicted using filled and hollow circles respectively.

We use the 3 learned cost functions and our pre-trained visual dynamics model to optimize a sequence of $T = 10$ desired joint angle displacements towards the keypoint goal from demonstraion. We record the mean squared distance to the goal keypoint in Table 1. We note that while both Time Dependent and the RBF Weighted Costs perform much better than our baseline, the simple Weighted Cost performs well on just one of the test cases, indicating that the time-dependency component of the cost leads to better generalization.

Start	**Weighted** Mean (Std)	**TimeDep** Mean (Std)	**RBF** Mean (Std)	**Default** Mean (Std)
1	40.99 (9.08)	6.12 (0.94)	**4.26** (**1.20**)	26.96 (5.41)
2	3.61 (0.40)	**3.53** (**0.21**)	4.40 (0.15)	15.76 (1.34)

Table 1: records the mean squared distance between the keypoints obtained after executing an action trajectory optimized from the indicated cost on the KUKA to the given goal keypoints from 2 starting configurations.

6 Discussion and Future Work

We propose a gradient-based IRL framework that learns cost function from visual human demonstrations. We learn a compact keypoint-based image representation, and train a visual dynamics in that latent space. We then use the keypoint trajectories extracted from user demonstrations, and our learned dynamics model, to learn different cost functions using our gradient-based IRL algorithm.

Several challenges remain: Learning a good visual predictive model is difficult, and created one of the main challenges in this work. One avenue for easier dynamics model training is to robustify the keypoint detector using methods like Florence et al. [27], so that it becomes invariant to different viewpoints. Furthermore, our work assumes that demonstrations are given from the perspective of the robot. We account for different starting configurations by learning on relative demonstrations instead of absolute. A step towards generalizing our approach even more is to consider methods that can map demonstrations from one context to another, as was presented in Liu et al. [28]. Finally, while we have presented experimental results for the more improved convergence behavior of our gradient-based IRL algorithm, as compared to the feature-matching baseline, we would like to investigate our findings in more depth in future work.

Acknowledgments

We would like to thank Kristen Morse for her useful suggestions during the preparation of this manuscript as well as Masoumeh Aminzadeh for discussions during the early stages of this project.

References

[1] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters. An algorithmic perspective on imitation learning. *arXiv preprint arXiv:1811.06711*, 2018.

[2] M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal. Learning objective functions for manipulation. In *2013 IEEE International Conference on Robotics and Automation*, pages 1331–1336, 2013.

[3] A. Boularias, J. Kober, and J. Peters. Relative entropy inverse reinforcement learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 182–189, 2011.

[4] C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58, 2016.

[5] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

[6] P. Englert, N. A. Vien, and M. Toussaint. Inverse kkt: Learning cost functions of manipulation tasks from demonstrations. *The International Journal of Robotics Research*, 36(13-14):1474–1488, 2017.

[7] M. Wulfmeier, D. Rao, D. Z. Wang, P. Ondruska, and I. Posner. Large-scale cost function learning for path planning using deep inverse reinforcement learning. *The International Journal of Robotics Research*, 36(10):1073–1087, 2017.

[8] E. Grefenstette, B. Amos, D. Yarats, P. M. Htut, A. Molchanov, F. Meier, D. Kiela, K. Cho, and S. Chintala. Generalized inner loop meta-learning. *arXiv preprint arXiv:1910.01727*, 2019.

[9] C. Finn and S. Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE, 2017.

[10] F. Ebert, C. Finn, S. Dasari, A. Xie, A. X. Lee, and S. Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *CoRR*, abs/1812.00568, 2018. URL http://arxiv.org/abs/1812.00568.

[11] R. Y. Rubinstein and D. P. Kroese. *The Cross Entropy Method: A Unified Approach To Combinatorial Optimization, Monte-Carlo Simulation (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2004. ISBN 038721240X.

[12] A. Byravan, F. Leeb, F. Meier, and D. Fox. Se3-pose-nets: Structured deep dynamics models for visuomotor control. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3339–3346, 2018.

[13] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pages 2746–2754, 2015.

[14] L. Manuelli, W. Gao, P. Florence, and R. Tedrake. kpam: Keypoint affordances for category-level robotic manipulation. *International Symposium on Robotics Research (ISRR)*, 2019.

[15] M. Minderer, C. Sun, R. Villegas, F. Cole, K. P. Murphy, and H. Lee. Unsupervised learning of object structure and dynamics from videos. In *Advances in Neural Information Processing Systems*, pages 92–102, 2019.

[16] T. D. Kulkarni, A. Gupta, C. Ionescu, S. Borgeaud, M. Reynolds, A. Zisserman, and V. Mnih. Unsupervised learning of object keypoints for perception and control. In *Advances in neural information processing systems*, pages 10724–10734, 2019.

[17] M. Lambeta, P. Chou, S. Tian, B. Yang, B. Maloon, V. R. Most, D. Stroud, R. Santos, A. Byagowi, G. Kammerer, D. Jayaraman, and R. Calandra. Digit: A novel design for a low-cost compact high-resolution tactile sensor with application to in-hand manipulation. *IEEE Robotics and Automation Letters*, 5(3):3838–3845, 2020.

[18] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[19] P. Abbeel, A. Coates, and A. Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[20] J. Zhao and L. Zhang. Inverse reinforcement learning with model predictive control. *Semantic Scholar*, 2019.

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

[22] S. Bechtle, A. Molchanov, Y. Chebotar, E. Grefenstette, L. Righetti, G. Sukhatme, and F. Meier. Meta-learning via learned loss. *arXiv preprint arXiv:1906.05374*, 2019.

[23] K. Lee, B. Vlahov, J. Gibson, J. M. Rehg, and E. A. Theodorou. Approximate inverse reinforcement learning from vision-based imitation learning. *arXiv preprint arXiv:2004.08051*, 2020.

[24] P. Sermanet, K. Xu, and S. Levine. Unsupervised perceptual rewards for imitation learning. *arXiv preprint arXiv:1612.06699*, 2016.

[25] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[26] D. Lee, S. Yoon, S. Lee, and G. Lee. Let's do inverse rl, 2019. URL https://github.com/reinforcement-learning-kr/lets-do-irl. [Online; accessed July 2020].

[27] P. R. Florence, L. Manuelli, and R. Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. In *Conference on Robot Learning*, pages 373–385, 2018.

[28] Y. Liu, A. Gupta, P. Abbeel, and S. Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1118–1125. IEEE, 2018.

A Appendix

Our framework consists of several components trained in isolation, eg the keypoint detector and the dynamics model. Here we describe the architecture and training details of both. Furthermore, we go into the details of our baseline implementation (A.4) as well as our visual MPC trajectory optimization step (A.5). Finally we also visually depict the results for the 2nd starting configuration of experiments done in Section 5.2 (A.6).

A.1 Keypoint Detector And Dynamics Model Architectures

Here we describe the architecture details of both the keypoint detector and the dynamics model.

keypoint detector g_{key}: The complete architecture for training the keypoint detector comprises of an autoencoder with a structural bottleneck that can extract "significant" 2D locations from the input images. g_{key} itself is essentially the encoder component of the autoencoder. Following Lambeta et al. [17], we implement g_{key} as a mini version of ResNet-18. The input images used are cropped to a resolution of $[240 \times 240]$.

dynamics model f_{dyn}: Our dynamics model $\hat{s}_t = f_{\text{dyn}}(\hat{s}_{t-1}, u_{t-1})$, where $\hat{s}_t = [\hat{z}_t, \hat{\theta}_t, \hat{\dot{\theta}}_t]$, has 2 components:

1) a keypoint predictor f_{mlp}

$$\hat{z}_t = f_{\text{mlp}}(\hat{s}_{t-1}, u_{t-1});$$

which is modeled by a neural network with two hidden layers with 100 and 25 neurons respectively and a ReLu activations after each layer except the last.

2) a next joint state predictor which simply integrates the action u_{t-1}, which are desired joint angle displacements, with the current (predicted) joint state $\hat{\theta}_{t-1}, \hat{\dot{\theta}}_{t-1}$ to predict the next state:

$$\hat{\theta}_t = \hat{\theta}_{t-1} + u_{t-1}$$
$$\hat{\dot{\theta}}_t = \hat{\dot{\theta}}_{t-1}$$

A.2 Self-Supervised Training of Keypoint detector

To train our keypoint detector we collect 108 sequences of video data, each 10 frames long. For each sequence we move the the Kuka iiwa while gripping an object into a random configuration, and then only move the last joint such that the detector emphasizes on extracting 2D locations that correspond to the gripped object as opposed to the robot arm. We train the keypoint detector until convergence. For additional details regarding the training process refer to Minderer et al. [15]. The resulting keypoint detector is visualized in Figure 3.

A.3 Training of Dynamics Model in Latent Space

We train 2 separate dynamics models f_{dyn} for the two sets of experiments.

Experiments of Section 5.1 For the first task of moving the bottle in 'x'-direction we use a purely self-supervised data collection routine. We command sine motions at various frequencies and amplitudes to each joint of the arm. The sine motions were designed to move joints 2,4 and 6 the most, such that the arm stays in plane. The trained keypoint detector is running asynchronously at $30hz$, and outputs z_t at that rate. We collect tuples (s_t, u_t, s_{t+1}), where $s_t = [\theta_t, \dot{\theta}_t, z_t]$, at a frequency of 5Hz, on which we train the dynamics model.

The parameters of f_{dyn} are trained by optimizing a normalized mean squared error (NMSE) between predicted $\hat{s}_{t+1} = f_{\text{dyn}}(s_t, u_t)$ and the ground truth s_{t+1}. We train this model until we converge to a NMSE of 0.3.

Experiments of Section 5.2 For the task of placing a bottle, we combine self-supervised data collection, with data collected from expert controllers (that roughly achieve the placing task), and data augmentation techniques. The self-supervised data collection is similar to the one described

above. The trained keypoint detector is running asynchronously at $30hz$, and outputs z_t at that rate. We collect tuples (s_t, u_t, s_{t+1}), where $s_t = [\theta_t, \dot{\theta}_t, z_t]$, at a frequency of 1Hz, to train the dynamics model.

The parameters of f_{dyn} are trained by optimizing a normalized mean squared error (NMSE) between predicted $\hat{s}_{t+1} = f_{\text{dyn}}(s_t, u_t)$ and the ground truth s_{t+1}. We were able to train this model to a NMSE of 0.03.

A.4 Adaptation of Abeel's IRL algorithm

We extend the publicly available implementation [26] for our baseline comparison of Abbeel and Ng [5]. We change their inner loop, to use our model-based trajectory optimisation routine. Further, we saw fair to use as features $\phi(\cdot)$ the per-step task objective $\mathscr{L}_{IRL}(\cdot)$ employed in this work. Finally, [26] had a larger value for the minimal distance from baseline constraint (2 instead of 1) than the one suggested in the original paper [5]. We found that using 1 as advised by the authors to work better than [26]. Therefore, the overall algorithm remains similar to the introduced by Abbeel and Ng [5] and namely,

Algorithm 2 Apprenticeship Learning Algorithm

Randomly initialise parameters $u(0)$ and pre-trained f_{dyn}, compute $\mu(0) = \mu(\pi(0))$.
demo τ_{demo}, with goal state $z_{\text{goal}} = \tau_T$
initial state $s_0 = (\theta_0, \dot{\theta}_0, z_0)$
for each *epoch* **do**
  $u_t = 0, \forall t = 1, \ldots, T$
  // Take maximal ψ by using the expectation of features from the final rollouts.
  $\mathscr{L}_{IRL} = max_{\psi:||\psi||_2 \leq 1} min_{j \in [0..(epoch-1)]} \psi^T(\mu_E - \mu(j))$
  let $\psi(epoch)$ be the value of ψ that attains this max.
  for each i in $\text{iters}_{\text{max}}$ **do**
    // rollout $\hat{\tau}$ from initial state s_0 and actions u
    $\hat{\tau} \leftarrow \text{rollout}(s_0, u, f_{\text{dyn}})$
    // Gradient descent on u with current $C_\psi = \psi^T \phi(\cdot)$
    $u_{new} \leftarrow u - \alpha . \nabla_u C_\psi(\hat{\tau}, \psi)$
  end for
  // Current expectation of features $\mu(\cdot)$ is over all features from the final rollout.
  Compute $\mu(\cdot) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t \phi(z_t)]$.
end for

A.5 Details regarding the Evaluation of different cost functions

We evaluate the baseline and learned cost functions by comparing the keypoint from the last step of planned trajectory they optimize with the goal keypoint. The planned trajectory is extracted using Algorithm 3. Our evaluation metric *relative distance* is defined as $\frac{||\tilde{z}_T - z_{goal}||}{||\tilde{z}_0 - z_{goal}||}$

Algorithm 3 Trajectory planning using given Cost

Given the cost C_ψ, planning horizon T, the forward dynamics model f_{dyn} and the initial state $s_0 = [z_0, \theta_0]$ .... where z_t, θ_t denote the keypoint and joint vector at time t and z_{goal} denotes the goal keypoint vector.

Initialize $u_{\text{init},t} = 0, \forall t = 1, \ldots, T$

// Rollout using the initial actions

$\hat{z}_0 = z_0, \hat{\theta}_0 = \theta_0$

$\hat{\tau} = \{\hat{z}_0\}$

for $t \leftarrow 1 : T$ **do**

  $\hat{s}_{t-1} = [\hat{z}_{t=1}, \hat{\theta}_{t-1}, \hat{\dot{\theta}}_{t-1}]^T$

  $\hat{z}_t, \hat{\theta}_t, \hat{\dot{\theta}}_t = f_{\text{dyn}}(\hat{s}_{t-1}, u_{\text{init},t-1})$

  $\hat{\tau} \leftarrow \hat{\tau} \cup \hat{z}_t$

end for

//Action optimization

$u_{\text{opt}} \leftarrow u_{\text{init}} - \alpha . \nabla_u C(\hat{\tau}, z_{goal})$

//Get planned trajectory by rolling out u_{opt}

$\tilde{z}_0 = z_0, \tilde{\theta}_0 = \theta_0, \tilde{\dot{\theta}}_0 = \dot{\theta}_0$

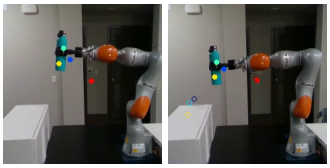
for $t \leftarrow 1 : T$ **do**

  $\tilde{z}_t, \tilde{\theta}_t = f_{\text{dyn}}([\tilde{z}_{t-1}, \tilde{\theta}_{t-1}, \tilde{\dot{\theta}}_{t-1}], u_{\text{opt},t-1})$

end for

Return $\tilde{z}, \tilde{\theta}$

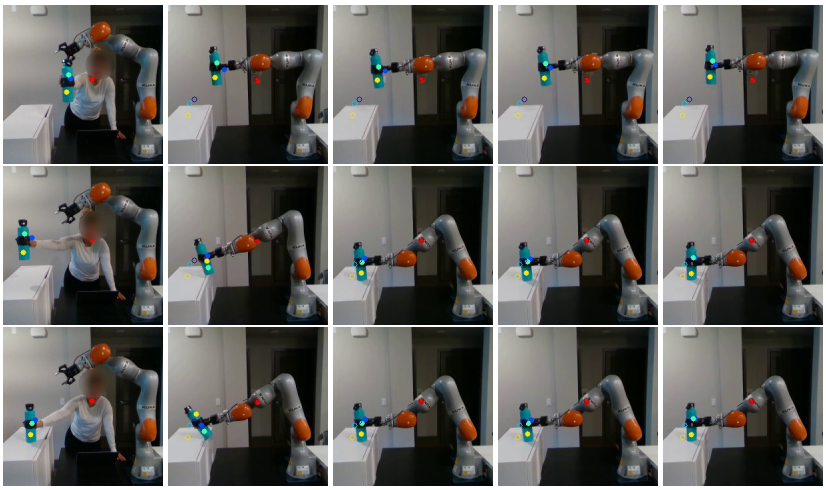
A.6 Additional Results



(a) Test Case 1 (b) Test Case 2

Figure 8: The 2 starting configurations for the placing task we evaluate our approach on.

Figure 9 visually depicts the results for starting configuration 2.



(a) Demo (b) Default (c) Weighted (d) TimeDep (e) RBF

Figure 9: Column a) Human Demonstration that is used for the IRL algorithm to extract cost functions. Column b)-d) Comparison of visual MPC result using the default and learned costs. First row corresponds to timestep $t = 0$, middle row to $t = 5$ and bottom row to $t = 10$ of executing the placing task. The detected and goal keypoints in each image are depicted using filled and hollow circles respectively.