# UNSUPERVISED PERCEPTUAL REWARDS FOR IMITATION LEARNING

**Pierre Sermanet, Kelvin Xu* & Sergey Levine**
Google Brain
{sermanet,kelvinxx,slevine}@google.com



## ABSTRACT

Reward function design and exploration time are arguably the biggest obstacles to the deployment of reinforcement learning (RL) agents in the real world. In many real-world tasks, designing a suitable reward function takes considerable manual engineering and often requires additional and potentially visible sensors to be installed just to measure whether the task has been executed successfully. Furthermore, many interesting tasks consist of multiple steps that must be executed in sequence. Even when the final outcome can be measured, it does not necessarily provide useful feedback on these implicit intermediate steps or sub-goals.

To address these issues, we propose leveraging the abstraction power of intermediate visual representations learned by deep models to quickly infer perceptual reward functions from small numbers of demonstrations. We present a method that is able to identify the key intermediate steps of a task from only a handful of demonstration sequences, and automatically identify the most discriminative features for identifying these steps. This method makes use of the features in a pre-trained deep model, but does not require any explicit sub-goal supervision. The resulting reward functions, which are dense and smooth, can then be used by an RL agent to learn to perform the task in real-world settings. To evaluate the learned reward functions, we present qualitative results on two real-world tasks and a quantitative evaluation against a human-designed reward function. We also demonstrate that our method can be used to learn a complex real-world door opening skill using a real robot, even when the demonstration used for reward learning is provided by a human using their own hand. To our knowledge, these are the first results showing that complex robotic manipulation skills can be learned directly and without supervised labels from a video of a human performing the task.

## 1 INTRODUCTION

Social learning, such as imitation, plays a critical role in allowing humans and animals to quickly acquire complex skills in the real world. Humans can use this weak form of supervision to acquire behaviors from very small numbers of demonstrations, in sharp contrast to deep reinforcement learning (RL) methods, which typically require extensive training data. In this work, we make use of two ideas about imitation to develop a scaleable and efficient imitation learning method: first, imitation makes use of extensive prior knowledge to quickly glean the "gist" of a new task from even a small number of demonstrations; second, imitation involves both observation and trial-and-error learning (RL). Building on these ideas, we propose a reward learning method for understanding the intent of a user demonstration through the use of pre-trained visual features, which provide the "prior knowledge" for efficient imitation. Our algorithm aims to discover not only the high-level goal of a task, but also the implicit sub-goals and steps that comprise more complex behaviors. Extracting such sub-goals can allow the agent to make maximal use of the information contained in the demonstration. Once the reward function has been extracted, the agent can use its own experience at the task to determine the physical structure of the behavior, even when the reward is provided by an agent with a substantially different embodiment (e.g. a human providing a demonstration for a robot).

*Work done as part of the Google Brain Residency program (g.co/brainresidency).

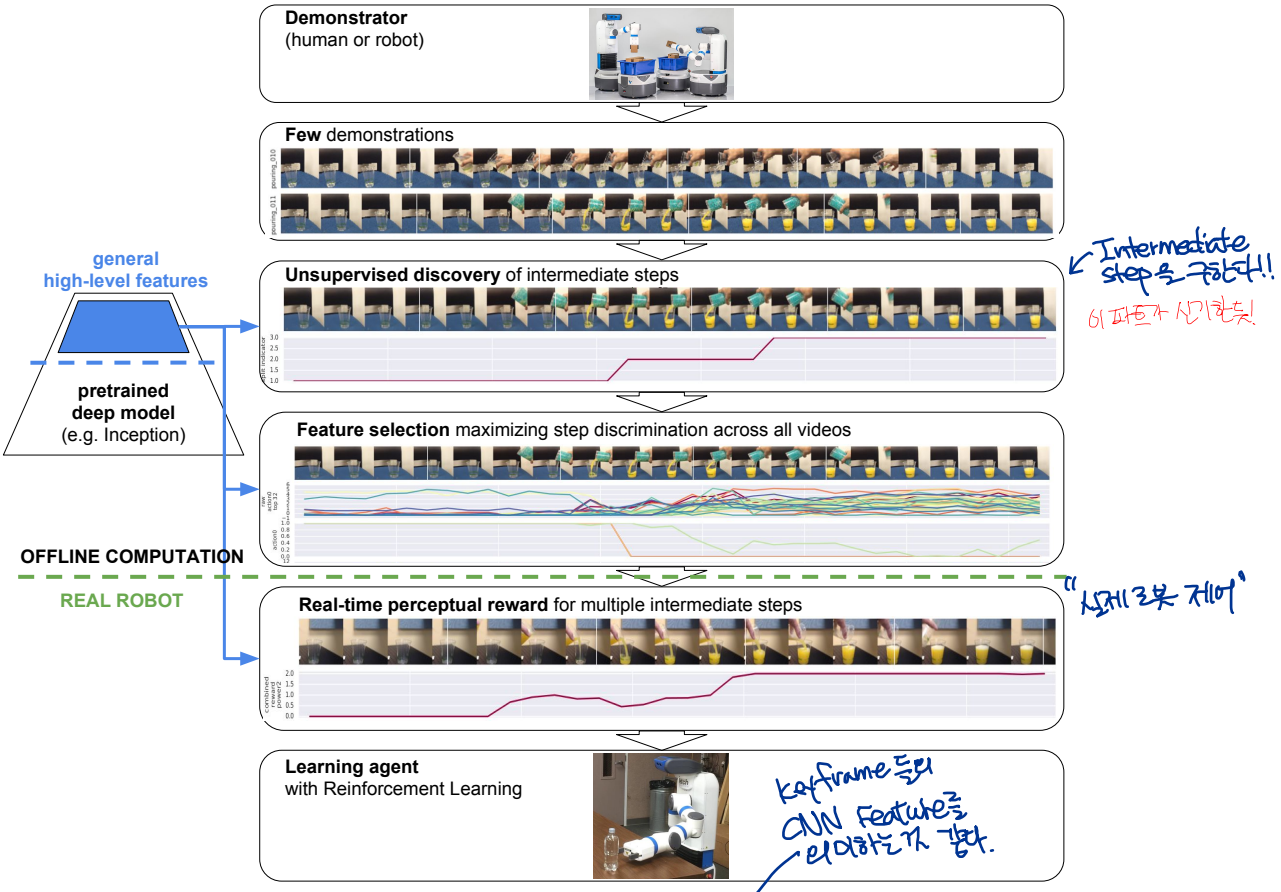


Figure 1: **Method overview.** Given a few demonstration videos of the same action, our method discovers intermediate steps, then selects for each step the most discriminative features found in the mid and high-level representations of a pre-trained deep model (in this work, we use all activations starting from the first "mixed" layer that follows the first 5 convolutional layers). The selected features are then combined to produce a single reward function per step. These intermediate rewards are combined into a single reward function. The reward function is then used by a real robot to learn the perform the demonstrated task as show in 3.2.

To our knowledge, our method is the first reward learning technique that learns generalizable vision-based reward functions for complex robotic manipulation skills from only a few demonstrations provided directly by a human. Although prior methods have demonstrated reward learning with vision for real-world robotic tasks, they have either required kinesthetic demonstrations with robot state for reward learning (Finn et al., 2015), or else required low-dimensional state spaces and numerous demonstrations (Wulfmeier et al., 2016). The contributions of this paper are:

- A method for perceptual reward learning from only a **few demonstrations** of **real-world** tasks. Reward functions are dense and incremental, with automated **unsupervised** discovery of intermediate steps.
- **The first vision-based reward learning method** that can learn a complex robotic manipulation task from a few human demonstrations in **real-world robotic experiments**.
- A set of empirical experiments that show that the learned visual representations inside a pre-trained deep model are general enough to be directly used to represent goals and subgoals for manipulation skills in new scenes **without retraining**.

### 1.1 Related Work

Deep reinforcement learning and deep robotic learning work has previously examined learning reward functions based on images. One of the most common approaches to image-based reward functions is to directly specify a "target image" by showing the learner the raw pixels of a successful task completion state, and then using distance to that image (or its latent representation) as a reward function (Lange et al., 2012; Finn et al., 2015; Watter et al., 2015). However, this approach

has several severe shortcomings. First, the use of a target image presupposes that the system can achieve a substantially similar visual state, which precludes generalization to semantically similar but visually distinct situations. Second, the use of a target image does not provide the learner with information about which facet of the image is more or less important for task success, which might result in the learner excessively emphasizing irrelevant factors of variation (such as the color of a door due to light and shadow) at the expense of relevant factors (such as whether or not the door is open or closed). Analyzing a collection of demonstrations to learn a parsimonious reward function that explains the demonstrated behavior is known as inverse reinforcement learning (IRL) (Ng et al., 2000). A few recently proposed IRL algorithms have sought to combine IRL with vision and deep network representations (Finn et al., 2016b; Wulfmeier et al., 2016). However, scaling IRL to high-dimensional systems and open-ended reward representations is very challenging. The previous work closest to ours used images together with robot state information (joint angles and end effector pose), with tens of demonstrations provided through kinesthetic teaching (Finn et al., 2016b). The approach we propose in this work which can be interpreted as a simple and efficient approximation to IRL, can use demonstrations that consist of videos of a human performing the task using their own body, and can acquire reward functions with intermediate sub-goals using just a few examples. This kind of efficient vision-based reward learning from videos of humans has not been demonstrated in prior IRL work. The idea of perceptual reward functions using raw pixels was also explored by Edwards et al. (2016) which, while sharing the same spirit as this work, was limited to simple synthetic tasks and used single images as perceptual goals rather than multiple demonstration videos.

## 2 Simple Inverse Reinforcement Learning with Visual Features

The key insight in our approach is that we can exploit the semantically meaningful and powerful features in a pre-trained deep neural network to infer task goals and sub-goals using a very simple approximate inverse reinforcement learning method. The pre-trained network effectively transfers prior knowledge about the visual world to make imitation learning fast and robust. Our approach can be interpreted as a simple approximation to inverse reinforcement learning under a particular choice of system dynamics, as discussed in Section 2.1. While this approximation is somewhat simplistic, it affords an efficient and scaleable learning rule that avoids overfitting even when trained on a small number of demonstrations. As depicted in Fig. 1, our algorithm first segments the demonstrations into segments based on perceptual similarity, as described in Section 2.2. Intuitively, the resulting segments correspond to sub-goals or steps of the task. The segments can then be used as a supervision signal for discriminative feature selection, described in Section 2.3, which produces a single perception reward function for each step of the task using only the most informative features, to further reduce overfitting. The combined reward function can then be used with a reinforcement learning algorithm to learn the demonstrated behavior. Although this method for extracting reward functions is exceedingly simple, its power comes from the use of highly general and robust pre-trained visual features, and our key empirical result is that such features are sufficient to acquire effective and generalizable reward functions for real-world manipulation skills.

We use the Inception network (Szegedy et al., 2015) pre-trained ImageNet classification (Deng et al., 2009) to obtain the visual features for representing the learned rewards. It is well known that visual features in such networks are quite general and can be reused for other visual tasks. However, it is less clear if sparse subsets of such features can be used directly to represent goals and sub-goals for real-world manipulation skills. Our experimental evaluation suggests that indeed they can, and that the resulting reward representations are robust and reliable enough for real-world robotic learning without any finetuning of the features. In this work, we use all activations starting from the first "mixed" layer that follows the first 5 convolutional layers (this layer's activation map is of size 35x35x256 given a 299x299 input). While this paper focuses on visual perception, the approach is general and can be applied to other modalities (e.g. audio and tactile).

### 2.1 Inverse Reinforcement Learning with Time-Independent Gaussian Models

Inverse reinforcement learning can be performed with a variety of algorithms (Ng et al., 2000), ranging from margin-based methods (Abbeel & Ng, 2004; Ratliff et al., 2006) to methods based on probabilistic models (Ramachandran & Amir, 2007; Ziebart et al., 2008). In this work, we use a very simple approximation to the MaxEnt IRL model (Ziebart et al., 2008), a popular probabilistic approach to IRL. We will use $s_t$ to denote the visual feature activations at time $t$, which constitute the state, $s_{it}$ to denote the $i^{\text{th}}$ feature at time $t$, and $\tau = \{s_1, \ldots, s_T\}$ to denote a sequence or trajectory

of these activations in a video. In MaxEnt IRL, the demonstrated trajectories $\tau$ are assumed to be drawn from a Boltzmann distribution according to:

$$p(\tau) = p(s_1, \ldots, s_T) = \frac{1}{Z} \exp\left(\sum_{t=1}^{T} R(s_t)\right), \quad (1)$$

where $R(s_t)$ is the unknown reward function. The principal computational challenge in MaxEnt IRL is to approximate $Z$, since the states at each time step are not independent, but are constrained by the system dynamics. In deterministic systems, where $s_{t+1} = f(s_t, a_t)$ for actions $a_t$ and dynamics $f$, the dynamics impose constraints on which trajectories $\tau$ are feasible. In stochastic systems, where $s_{t+1} \sum p(s_{t+1}|s_t, a_t)$, we must also account for the dynamics distribution in Equation (1), as discussed by Ziebart et al. (2008). Prior work has addressed this using dynamic programming to exactly compute $Z$ in small, discrete systems (Ziebart et al., 2008), or by using sampling to estimate $Z$ for large, continuous domains (Kalakrishnan et al., 2010; Boularias et al., 2011; Finn et al., 2016b). Since our state representation corresponds to a large vector of visual features, exact dynamic programming is infeasible. Sample-based approximation requires running a large number of trials to estimate $Z$ and, as shown in recent work (Finn et al., 2016a), corresponds to a variant of generative adversarial networks (GANs), with all of the accompanying stability and optimization challenges. Furthermore, the corresponding model for the reward function is complex, making it prone to overfitting when only a small number of demonstrations is available.

When faced with a difficult learning problem in extremely low-data regimes, a standard solution is to resort to simple, biased models, so as to minimize overfitting. We adopt precisely this approach in our work: instead of approximating the complex posterior distribution over trajectories under nonlinear dynamics, we use a simple biased model that affords efficient learning and minimizes overfitting. Specifically, we assume that all trajectories are dynamically feasible, and that the distribution over each activation at each time step is independent of all other activations and all other time steps. This corresponds to the IRL equivalent of a naïve Bayes model: in the same way that naïve Bayes uses an independence assumption to mitigate overfitting in high-dimensional feature spaces, we use independence between both time steps and features to learn from very small numbers of demonstrations. Under this assumption, the probability of a trajectory $\tau$ factorizes according to

$$p(\tau) = \prod_{t=1}^{T} \prod_{i=1}^{N} p(s_{it}) = \prod_{t=1}^{T} \prod_{i=1}^{N} \frac{1}{Z_{it}} \exp\left(R_i(s_{it})\right),$$

which corresponds to a reward function of the form $R_t(s_t) = \sum_{i=1}^{N} R_i(s_i t)$. We can then simply choose a form for $R_i(s_{it})$ that can be normalized analytically, which in our case is a quadratic in $s_{it}$, such that $\exp(R_i(s_{it}))/Z_{it}$ is a Gaussian distribution, and the original trajectory distribution is a naïve Bayes model. While this approximation is quite drastic, it yields an exceedingly simple learning rule: in the most basic version, we have only to fit the mean and variance of each feature distribution, and then use the log of the resulting Gaussian as the reward. Since the fit is performed independently for each feature, overfitting is minimized in comparison to more expressive models.

### 2.2 Discovery of Intermediate Steps

The simple IRL model in the previous section can be used to acquire a single quadratic reward function in terms of the visual features $s_t$. However, for complex multi-stage tasks, this model can be too coarse, making task learning slow and difficult. We therefore instead fit multiple quadratic reward functions, with one reward function per intermediate step or goal. These steps are discovered automatically in the first stage of our method, which is performed independently on each demonstration. If multiple demonstrations are available, they are pooled together in the feature selection step discussed in the next section, and could in principle be combined at the segmentation stage as well, though we found this to be unnecessary in our prototype. The intermediate steps model extends the simple independent Gaussian model in the previous section by assuming that

$$p(\tau) = \prod_{t=1}^{T} \prod_{i=1}^{N} \frac{1}{Z_{it}} \exp\left(R_{i g_t}(s_{it})\right),$$

where $g_t$ is the index of the goal or step corresponding to time step $t$. Learning then corresponds to identifying the boundaries of the steps in the demonstration, and fitting independent Gaussian feature

distributions at each step. Note that this corresponds exactly to segmenting the demonstration such that the variance of each feature within each segment is minimized.

The goal of the segmentation phase is therefore to find the splitting points in each sequence that minimize the average features variance across all segments. This can be implemented using the simple recursive method in Algorithm 1. Intuitively, this method breaks down a sequence in a way that each frame in a segment is abstractly similar to each other frame in that segment. The number of segments is provided manually in this approach, though it would be straightforward to also utilize standard model selection criteria for choosing this number automatically. The complexity of Algorithm 1 is $\mathcal{O}(n^m)$ where $n$ is the number of frames in a sequence and $m$ the number of splits. Note that dynamic programming is not applicable to this algorithm because each sub-problem, i.e. how to split a sequence after the $i^{th}$ frame, depends on the segmentation chosen before the $i^{th}$ frame. We also experiment with a greedy binary version of this algorithm (Algorithm 2 detailed in section A.1): first split the entire sequence in two, then recursively split each new segment in two. While not exactly minimizing the variance across all segments, it is significantly more efficient ($\mathcal{O}(n^2 \log m)$) and yields qualitatively sensible results.

**Algorithm 1** Recursive similarity maximization, where $AverageStd()$ is a function that computes the average standard deviation over a set of frames or over a set of values, $Join()$ is a function that joins values or lists together into a single list, $n$ is the number of splits desired and $min\_size$ is the minimum size of a split.

```
function SPLIT(video, start, end, n, min_size, prev_std = [])
    if n = 1 then return [], [AVERAGESTD(video[start : end])]
    end if
    min_std ← None
    min_std_list ← []
    min_split ← []
    for i ← start + min_size to end − ((n − 1) ∗ min_size)) do
        std1 ← [AVERAGESTD(video[start : i])]
        splits2, std2 ← SPLIT(video, i, end, n − 1, min_size, prev_std + std1)
        avg_std ← AVERAGESTD(JOIN(prev_std, std1, std2))
        if min_std = None or avg_std < min_std then
            min_std ← avg_std
            min_std_list ← JOIN(std1, std2)
            min_split ← JOIN(i, splits2)
        end if
    end for
    return min_split, min_std_list
end function
```

### 2.3 FEATURE SELECTION

Although we could directly use the feature means and variances of each segment identified via Algorithm 1, in practice we can reduce overfitting even further by retaining only the most discriminative features for each segment. To that end, the second stage in our method aims to select the most relevant features for each segmented step or goal.

Intent understanding requires identifying highly discriminative features of a specific goal while remaining invariant to unrelated variation (e.g. lighting, color, viewpoint). The relevant discriminative features may be very diverse and more or less abstract, which motivates our intuition to tap into the activations of deep models at different depths. Deep models cover a large set of representations that can be useful, from spatially dense and simple features in the lower layers (e.g. large collection of detected edges) to gradually more spatially sparse and abstract features (e.g. few object classes). We hypothesize that in mid to high-level features, there exists enough sparse independent features that can readily and compactly discriminate between a wide range of previously unseen inputs.

To select the most discriminative features, we use a simple scoring heuristic. Each feature $i$ is first normalized by subtracting the mean and dividing by the standard deviation of all training sequences. We then rank them for each sub-goal according to their distance $z_i$ to the average statistics of the sets of positive and negative frames for a given goal:

$$z_i = \alpha \left| \mu_i^+ - \mu_i^- \right| - (\sigma_i^+ + \sigma_i^-), \tag{2}$$

where $\mu_i^+$ and $\sigma_i^+$ are the mean and standard deviation of all "positive" frames and the $\mu_i^-$ and $\sigma_i^-$ of all "negative" frames (the frames that do not contain the sub-goal). Only the top-$M$ features are retained to form the reward function $R_g()$ for the sub-goal $g$, which is given by the log-probability of an independent Gaussian distribution over the relevant features:

$$R_g(s_t) = \frac{1}{n} \sum_{j}^{M} \frac{(s_{i_j t} - \mu_{i_j t}^+)^2}{{\sigma_{i_j t}^+}^2}, \tag{3}$$

where $i_j$ indexes the top-$M$ selected features. We empirically choose $\alpha = 5.0$ and $M = 32$ for our subsequent experiments. At test time, we do not know when the system transitions from one goal to another, so instead of time-indexing the goals, we instead combine all of the goals into a single time-invariant reward function, where later steps yield higher reward than earlier steps, as described in Appendix A.2.

### 2.4 Using Perceptual Rewards for Robotic Learning

In order to use our learned perceptual reward functions in a complete skill learning system, we must also choose a reinforcement learning algorithm and a policy representation. While in principle any reinforcement learning algorithm could be suitable for this task, we chose a method that is efficient enough to evaluate on real-world robotic systems in order to validate our approach. The method we use is based on the PI$^2$ reinforcement learning algorithm (Theodorou et al., 2010). Our implementation, which is discussed in more detail in Appendix A.3, uses a relatively simple linear-Gaussian parameterization of the policy, which corresponds to a sequence of open-loop torque commands with fixed linear feedback to correct for perturbations. This method also requires initialization from example demonstrations to learn complex manipulation tasks efficiently. A more complex neural network policy could also be used (Chebotar et al., 2016), and more sophisticated RL algorithms could also learn skills without demonstration initialization. However, since the main purpose of this component is to validate the learned reward functions, we used this simple approach to test our rewards quickly and efficiently.

Path Integral Guided Policy Search

## 3 Experiments

In this section, we discuss our empirical evaluation, starting with an analysis of the learned reward functions in terms of both qualitative reward structure and quantitative segmentation accuracy. We then present results for a real-world validation of our method on robotic door opening.

### 3.1 Perceptual Rewards Evaluation

We report results on two demonstrated tasks: door opening and liquid pouring. We collected about a dozen training videos for each task using a smart phone. As an example, Fig. 2 shows the entire training set used for the pouring task.

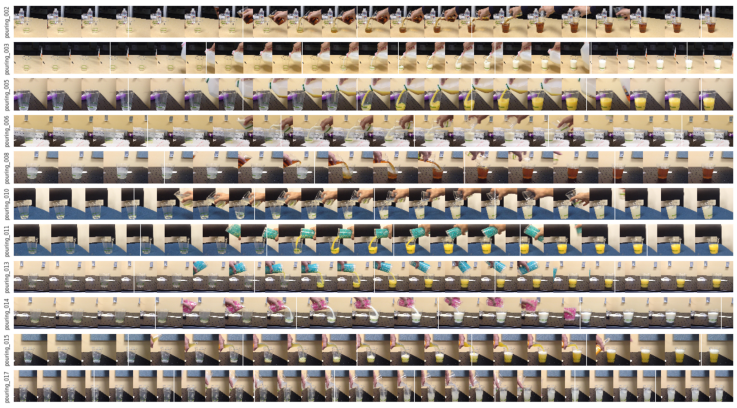

Figure 2: **Entire training set for the pouring task** (11 demonstrations).

### 3.1.1 QUALITATIVE ANALYSIS

While a door opening sensor can be engineered using sensors hidden in the door, measuring pouring or container tilting would be quite complicated, would visually alter the scene, and is unrealistic for learning in the wild. Visual reward functions are therefore an excellent choice for complex physical phenomena such as liquid pouring. In Fig. 3, we present the combined reward functions for test videos on the pouring task, and Fig. 9 shows the intermediate rewards for each sub-goal. We plot the predicted reward functions for both successful and failed task executions in Fig. 10. We observe that for "missed" executions where the task is only partially performed, the intermediate steps are correctly classified. Fig. 8 details qualitative results of unsupervised step segmentation for the door opening and pouring tasks. For the door task, the 2-segments splits are often quite in line with what one can expect, while a 3-segments split is less accurate. We also observe that the method is robust to the presence or absence of the handle on the door, as well as its opening direction. We find that for the pouring task, the 4-segments split often yields the most sensible break down. It is interesting to note that the 2-segment split usually occurs when the glass is about half full.

**Failure Cases**

The intermediate reward function for the door opening task which corresponds to a human hand manipulating the door handle seems rather noisy or wrong in 9b, 9c and 9e ("action1" on the y-axis of the plots).The reward function in 10f remains flat while liquid is being poured into the glass. The liquid being somewhat transparent, we suspect that it looks too similar to the transparent glass for the function to fire.

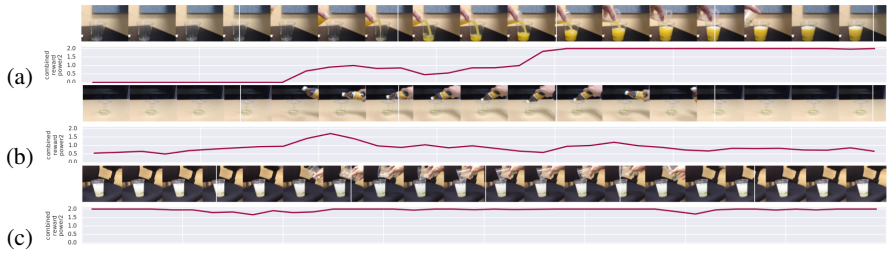


Figure 3: **Examples of "pouring" reward functions.** We show here a few successful examples, see Fig. 10 for results on the entire test set. In 3a we observe a continuous and incremental reward as the task progresses and saturating as it is completed. 3b increases as the bottle appears but successfully detects that the task is not completed, while in 3c it successfully detects that the action is already completed from the start.

### 3.1.2 QUANTITATIVE ANALYSIS

We evaluate the quantitative accuracy of the unsupervised steps discovery in Table 1, while Table 2 presents quantitative generalization results for the learned reward on a test video of each task. For each video, ground truth intermediate steps were provided by human supervision for the purpose of evaluation. While this ground truth is subjective, since each task can be broken down in multiple ways, it is reasonably consistent for the simple tasks in our experiments. We use the Jaccard similarity measure (intersection over union) to indicate how much a detected step overlaps with its corresponding ground truth.

Table 1: **Unsupervised steps discovery accuracy** (Jaccard overlap on training sets) versus the ordered random steps baseline.

| **dataset** | **method** | **2 steps** | | | **3 steps** | | | |
|---|---|---|---|---|---|---|---|---|
| (training) | | step 1 | step 2 | average | step 1 | step 2 | step 3 | average |
| door | ordered random steps | 59.4% | 45.6% | 52.5% | 48.0% | 58.1% | 60.1% | 55.4% |
| | unsupervised steps | **84.0%** | **68.1%** | **76.1%** | **57.6%** | **75.1%** | **68.1%** | **66.9%** |
| pouring | ordered random steps | 65.2% | 66.6% | 65.9% | 46.2% | 46.3% | **66.3%** | 52.9% |
| | unsupervised steps | **92.3%** | **90.5%** | **91.6%** | **79.7%** | **48.0%** | 48.6% | **58.8%** |

In Table 1, we compare our method against a random baseline. Because we assume the same step order in all demonstrations, we also order the random steps in time to provide a fair baseline. Note that the random baseline performs fairly well because the steps are distributed somewhat uniformly in

time. Should the steps be much less temporally uniform, the random baseline would be expected to perform very poorly, while our method should maintain similar performance. We compare splitting between 2 and 3 steps and find that, for both tasks, 2 steps are easier to discover, probably because these tasks exhibit one strong visual change each while the other steps are more subtle. Note that our unsupervised segmentation only works when full sequences are available while our learned reward functions can be used in real-time without accessing future frames. Hence in these experiments we evaluate the unsupervised segmentation on the training set only and evaluate the reward functions on the test set.

Table 2: **Reward functions accuracy** by steps (Jaccard overlap on test sets).

| **dataset** (testing) | **method** | **2 steps** | | | **3 steps** | | | |
|---|---|---|---|---|---|---|---|---|
| | | step 1 | step 2 | average | step 1 | step 2 | step 3 | average |
| door | random rewards | 41.5% | 23.4% | 32.4% | 21.5% | 32.9% | 25.4% | 26.6% |
| | learned rewards | **85.1%** | **59.7%** | **72.4%** | **56.9%** | **47.7%** | **54.1%** | **52.9%** |
| pouring | random reward | 40.9% | 25.1% | 33.0% | 22.0% | 39.4% | 14.0% | 25.2% |
| | learned rewards | **76.2%** | **54.6%** | **65.4%** | **32.9%** | **55.2%** | **32.2%** | **40.0%** |

In Table 2, we evaluate the reward functions individually for each step on the test set. For that purpose, we binarize the reward function using a threshold of $0.5$. The random baseline simply outputs true or false at each timestep. We observe that the learned rewards outperform the baseline by about a factor of 2. It is not clear exactly what level of accuracy is required to successfully learn to perform these tasks, but we show in section 3.2.2 that the reward accuracy on the door task is sufficient to reach $100\%$ success rate with a real robot.

### 3.2 Real-World Robotic Door Opening

In this section, we aim to answer the question of whether our previously visualized reward function can be used to learn a real-world robotic motion skill. We experiment on a door opening skill, where we adapt a demonstrated door opening to a novel configuration, such as different position or orientation of the door. Following the experimental protocol in prior work (Chebotar et al., 2016), we adapt an imperfect kinesthetic demonstration which we ensure succeeds at least occasionally (about 10% of the time). These demonstrations consist only of robot poses, and do not include images. We then use a variety of different video demonstrations, which contain images but not robot poses, to learn the reward function. These videos include demonstrations with other doors, and even demonstrations provided by a human using their own body, rather than through kinesthetic teaching with the robot.

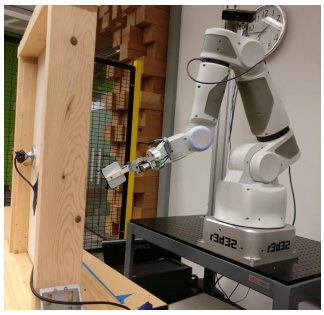

Figure 4: **Robot arm setup.** Note that our method does not make use of the sensor on the back handle of the door, but it is used in our comparison to train a baseline method with the ground truth reward.

Figure 4 shows the experimental setup. We use a 7-DoF robotic arm with a two-finger gripper, and a camera placed above the shoulder, which provides monocular RGB images. For our baseline PI$^2$ policy, we closely follow the setup of Chebotar et al. (2016) which uses an IMU sensor in the door handle to provide both a cost and feedback as part of the state of the controller. In contrast, in our approach we remove this sensor both from the state representation provided to PI$^2$ and in our reward replace the target IMU state with the output of a deep neural network.

#### 3.2.1 Data

We experiment with a range of different demonstrations from which we derive our reward function, varying both the source demo (human vs robotic), the number of subgoals we extract, and the appearance of the door. We record monocular RGB images on a camera placed above the shoulder of the arm. The door is cropped from the images, and then the resulting image is re-sized such that the shortest side is 299 dimensional with preserved aspect ratio. The input into our convolutional feature extractor Szegedy et al. (2015) is the 299x299 center crop.

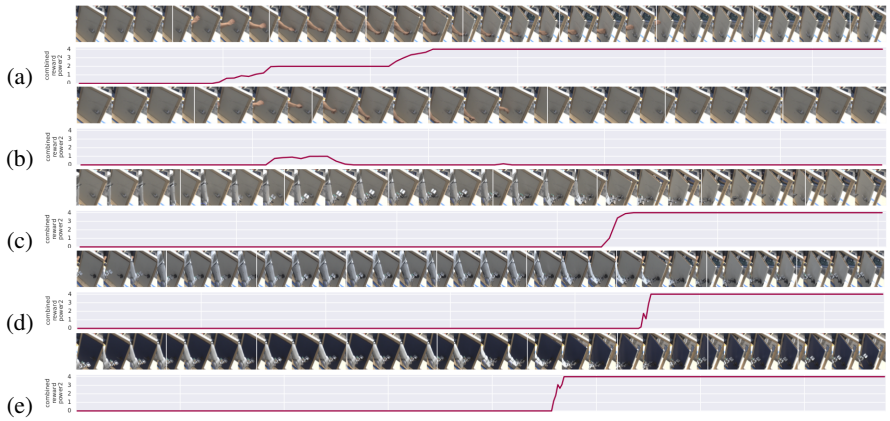


Figure 5: **Rewards from human demonstration only.** Here we show the rewards produced when trained on humans only (see Fig. 11). In 5a, we show the reward on a human test video. In 5b, we show what the reward produces when the human hands misses opening the door. In 5c, we show the reward successfully saturates when the robot opens the door even though it has not seen a robot arm before. Similarly in 5d and 5e we show it still works with some amount of variation of the door which was not seen during training (white door and black handle, blue door, rotations of the door).

### 3.2.2 Qualitative Analysis

We evaluate our reward functions qualitatively by plotting our perceptual reward functions below the demonstrations with a variety of door types and demonstrators (e.g robot or human). As can be seen in Fig. 5 and in real experiments Fig. 6, we show that the reward functions are useful to a robotic arm while only showing human demonstrations as depicted in Fig. 11. Moreover we exhibit robustness variations in appearance.

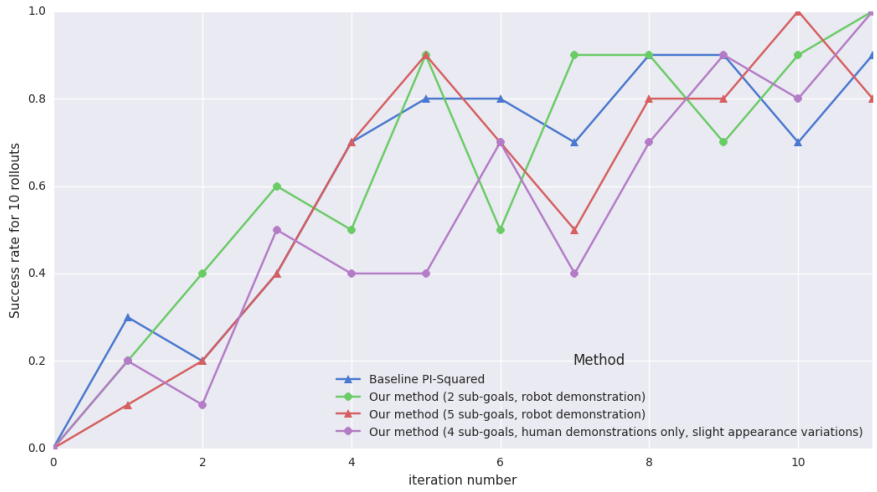


Figure 6: **Door opening success rate at each iteration of learning on the real robot.** The $PI^2$ baseline method uses a ground truth reward function obtained by instrumenting the door. Note that rewards learned by our method, even from videos of humans or different doors, learn comparably or faster when compared to the ground truth reward.

### 3.2.3 Quantitative Analysis

In comparing the success rate of visual reward versus a baseline $PI^2$ method that uses the ground truth reward function obtained by instrumenting the door with an IMU. We run $PI^2$ for 11 iterations

with 10 sampled trajectories at each iteration. As can be seen in Fig. 6, we obtain similar convergence speeds to our baseline model, with our method also able to open the door consistently. Since our local policy is able to obtain high reward candidate trajectories, this is strong evidence that a perceptual reward could be used to train a global in same manner as Chebotar et al. (2016).

## 4 CONCLUSION

In this paper, we present a method for automatically identifying important intermediate goal given a few visual demonstrations of a task. By leveraging the general features learned from pretrained deep models, we propose a method for rapidly learning an incremental reward function from human demonstrations which we successfully demonstrate on a real robotic learning task.
A compelling direction for future work is to explore how reward learning algorithms can be combined with robotic lifelong learning. One of the biggest barriers for lifelong learning in the real world is the ability of an agent to obtain reward supervision, without which no learning is possible. Continuous learning using unsupervised rewards promises to substantially increase the variety and diversity of experience that is available for robotic reinforcement learning, resulting in more powerful, robust, and general robotic skills.

### ACKNOWLEDGMENTS

We would like to thank Vincent Vanhoucke for helpful discussions and feedback. We would also like to thank Mrinal Kalakrishnan and Ali Yahya for indispensable guidance throughout this project.

## REFERENCES

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 1. ACM, 2004.

Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards AI. In *Large Scale Kernel Machines*. MIT Press, 2007.

Abdeslam Boularias, Jens Kober, and Jan Peters. Relative entropy inverse reinforcement learning. 2011.

Yevgen Chebotar, Mrinal Kalakrishnan, Ali Yahya, Adrian Li, Stefan Schaal, and Sergey Levine. Path integral guided policy search. *arXiv preprint arXiv:1610.00529*, 2016.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Ashley Edwards, Charles Isbell, and Atsuo Takanishi. Perceptual reward functions. *arXiv preprint arXiv:1608.03824*, 2016.

Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. *arXiv preprint arXiv:1509.06113*, 2015.

Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*, 2016a.

Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. *arXiv preprint arXiv:1603.00448*, 2016b.

Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

Mrinal Kalakrishnan, Evangelos Theodorou, and Stefan Schaal. Inverse reinforcement learning with pi 2. 2010.

Sascha Lange, Martin Riedmiller, and Arne Voigtländer. Autonomous reinforcement learning on raw visual input data in a real world application. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2012.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pp. 663–670, 2000.

Jan Peters, Katharina Mülling, and Yasemin Altün. Relative entropy policy search. In *AAAI Conference on Artificial Intelligence (AAAI 2010)*, 2010.

Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. *Urbana*, 51 (61801):1–4, 2007.

Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pp. 729–736. ACM, 2006.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. URL `http://arxiv.org/abs/1512.00567`.

Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research*, 11(Nov):3137–3181, 2010.

Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems*, pp. 2746–2754, 2015.

Markus Wulfmeier, Dominic Zeng Wang, and Ingmar Posner. Watch This: Scalable Cost-Function Learning for Path Planning in Urban Environments . In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016. arxiv preprint: http://arxiv.org/abs/1607.02329.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, pp. 1433–1438, 2008.

# A Algorithms Details

## A.1 Binary Segmentation Algorithm

## A.2 Combining Intermediate Rewards

From the two previous sections, we obtain one reward function per intermediate step discovered by the unsupervised algorithm. These need to be combined so that the RL algorithm uses a single reward function which partially rewards intermediate steps but most rewards the final one. The initial step is ignored as it is assumed to be the resting starting state in the demonstrations. We opt for the maximum range of each reward be twice the maximum range of its preceding reward, and summing them as follow:

$$R(a) = \sum_{i=2}^{n} R_i(a) * 2^{(i-1)} \tag{4}$$

where $n$ is the number of intermediate rewards detected and $a$ an activations vector. An example of this combination is shown in Fig. 7.

## A.3 PI$^2$ Reinforcement Learning Algorithm

We chose the PI$^2$ reinforcement learning algorithm (Theodorou et al., 2010) for our experiments, with the particular implementation of the method based on a recently proposed deep reinforcement learning variant (Chebotar et al., 2016). Since our aim is mainly to validate that our learned reward functions capture the goals of the task well enough for learning, we employ a relatively simple linear-Gaussian parameterization of the policy, which corresponds to a sequence of open-loop torque commands with fixed linear feedback to correct for perturbations, as in the work of Chebotar et al.

**Algorithm 2** Greedy and binary algorithm similar to and utilizing Algorithm 1, where $AverageStd()$ is a function that computes the average standard deviation over a set of frames or over a set of values, $Join()$ is a function that joins values or lists together into a single list, $n$ is the number of splits desired and $min\_size$ is the minimum size of a split.

**function** BINARYSPLIT($video, start, end, n, min\_size, prev\_std = []$)
  **if** $n = 1$ **then return** $[], []$
  **end if**
  $splits0, std0 \leftarrow$ SPLIT$(video, start, end, 2, min_size)$
  **if** $n = 2$ **then return** $splits0, std0$
  **end if**
  $splits1, std1 \leftarrow$ BINARYSPLIT$(video, start, splits0[0],$ CEIL$(n/2), min_size)$
  $splits2, std2 \leftarrow$ BINARYSPLIT$(video, splits0[0] + 1, end,$ FLOOR$(n/2), min_size)$
  $all\_splits = []$
  $all\_std = []$
  **if** $splits1 \neq []$ **then**
    JOIN$(all_splits, splits1)$
    JOIN$(all_std, std1)$
  **else**
    JOIN$(all_std, std0[0])$
  **end if**
  **if** $splits0 \neq []$ **then**
    JOIN$(all_splits, splits0[0])$
  **end if**
  **if** $splits2 \neq []$ **then**
    JOIN$(all_splits, splits2)$
    JOIN$(all_std, std2)$
  **else**
    JOIN$(all_std, std0[1])$
  **end if**
  **return** $all\_splits, all\_std$
**end function**

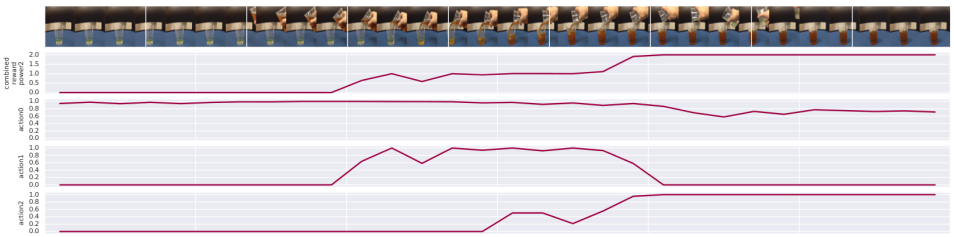

Figure 7: **Combining intermediate rewards into a single reward function.** From top to bottom, we show the combined reward function (with range [0,2]) followed by the reward function of each individual steps (with range [0,1]). The first step reward corresponds to the initial resting state of the demonstration and is ignored in the reward function. The second step corresponds to the pouring action and the third step corresponds to the glass full of liquid.

(2016). This policy has the form $\pi(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \Sigma_t)$, where $\mathbf{K}_t$ is a fixed stabilizing feedback matrix, and $\mathbf{k}_t$ is a learned control. In this case, the state $\mathbf{x}_t$ corresponds to the joint angles and angular velocities of a robot, and $\mathbf{u}_t$ corresponds to the joint torques. Since the reward function is evaluated from camera images, we assume that the image is a (potentially stochastic) consequence of the robot's state, so that we can evaluate the state reward $r(\mathbf{x}_t)$ by taking the image $\mathbf{I}_t$ observed at time $t$, and computing the corresponding activations $a_t$. Overloading the notation, we can write $a_t = f(\mathbf{I}_t(\mathbf{x}_t))$, where $f$ is the network we use for visual features. Then, we have $r(\mathbf{x}_t) = R(f(\mathbf{I}_t(\mathbf{x}_t)))$.

The PI$^2$ algorithm is an episodic policy improvement algorithm that uses the reward $r(\mathbf{x}_t)$ to iteratively improve the policy. The trust-region variant of PI$^2$ that we use Chebotar et al. (2016), which is also similar to the REPS algorithm (Peters et al., 2010), updates the policy at iteration $n$ by sampling from the time-varying linear-Gaussian policy $\pi(\mathbf{u}_t|\mathbf{x}_t)$ to obtain samples $\{(\mathbf{x}_t^{(i)}, \mathbf{u}_t^{(i)})\}$, and

updating the controls $\mathbf{k}_t$ at each time step according to

$$\mathbf{k}_t \leftarrow \left[\sum_i \mathbf{u}_t^{(i)} \exp\left(\beta_t \sum_{t'=t}^{T} r(\mathbf{x}_{t'}^{(i)})\right)\right] / \left[\sum_i \exp\left(\beta_t \sum_{t'=t}^{T} r(\mathbf{x}_{t'}^{(i)})\right)\right],$$

where the temperature $\beta_t$ is chosen to bound the KL-divergence between the new policy $\pi(\mathbf{u}_t|\mathbf{x}_t)$ and the previous policy $\bar{\pi}(\mathbf{u}_t|\mathbf{x}_t)$, such that $D_{\mathrm{KL}}(\pi(\mathbf{u}_t|\mathbf{x}_t)\|\bar{\pi}(\mathbf{u}_t|\mathbf{x}_t)) \leq \epsilon$ for a step size epsilon. Further details and a complete derivation are provided in prior work Theodorou et al. (2010); Peters et al. (2010); Chebotar et al. (2016).

The PI$^2$ algorithm is a local policy search method that performs best when provided with demonstrations to bootstrap the policy. In our experiments, we use this method together with our learned reward functions to learn a door opening skill with a real physical robot, as discussed in Section 3.2. Demonstration are provided with kinesthetic teaching, which results in a sequence of reference steps $\hat{\mathbf{x}}_t$, and initial controls $\mathbf{k}_t$ are given by $\mathbf{k}_t = -\mathbf{K}_t\hat{\mathbf{x}}_t$, such that the mean of the initial controller is $\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t)$, corresponding to a trajectory-following initialization. This initial controller is rarely successful consistently, but the occasional successes it achieves provide a learning signal to the algorithm. The use of demonstrations enables PI$^2$ to be used to quickly and efficiently learn complex robotic manipulation skills.

Although this particular RL algorithm requires demonstrations to begin learning, it can still provide a useful starting point for real-world learning with a real robotic system. As shown by Chebotar et al. (2016), the initial set of demonstrations can be expanded into a generalizable policy by iteratively "growing" the effective region where the policy succeeds. For example, if the robot is provided with a demonstration of opening a door in one position, additional learning can expand the policy to succeed in nearby positions, and the application of a suitable curriculum can grow the region of door poses in which the policy succeeds progressively. However, as with all RL algorithms, this process requires knowledge of the reward function. Using the method described in this paper, we can learn such a reward function from either the initial demonstrations or even from other demonstration videos provided by a human. Armed with this learned reward function, the robot could continue to improve its policy through real-world experience, iteratively increasing its region of competence through lifelong learning.

## B Additional Qualitative Results

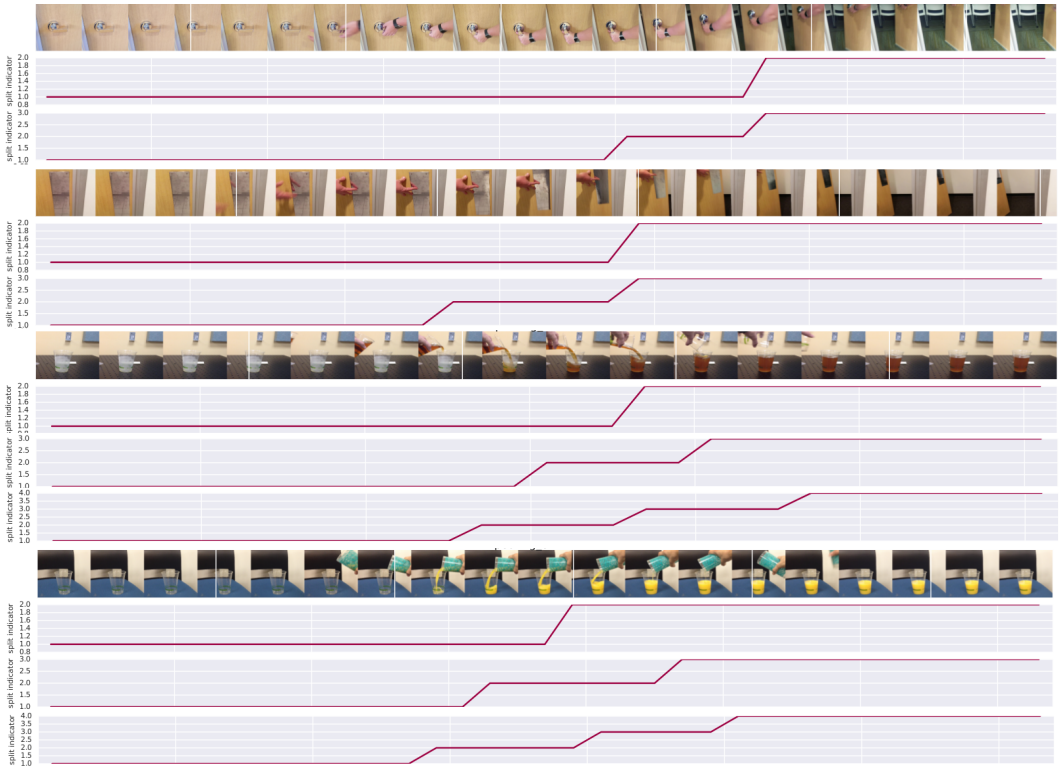

Figure 8: **Qualitative examples of unsupervised discovery of steps for door and pouring tasks** in training videos. For each video, we show the detected splits when splitting in 2, 3 or 4 segments. Each segment is delimited by a different value on the vertical axis of the curves.

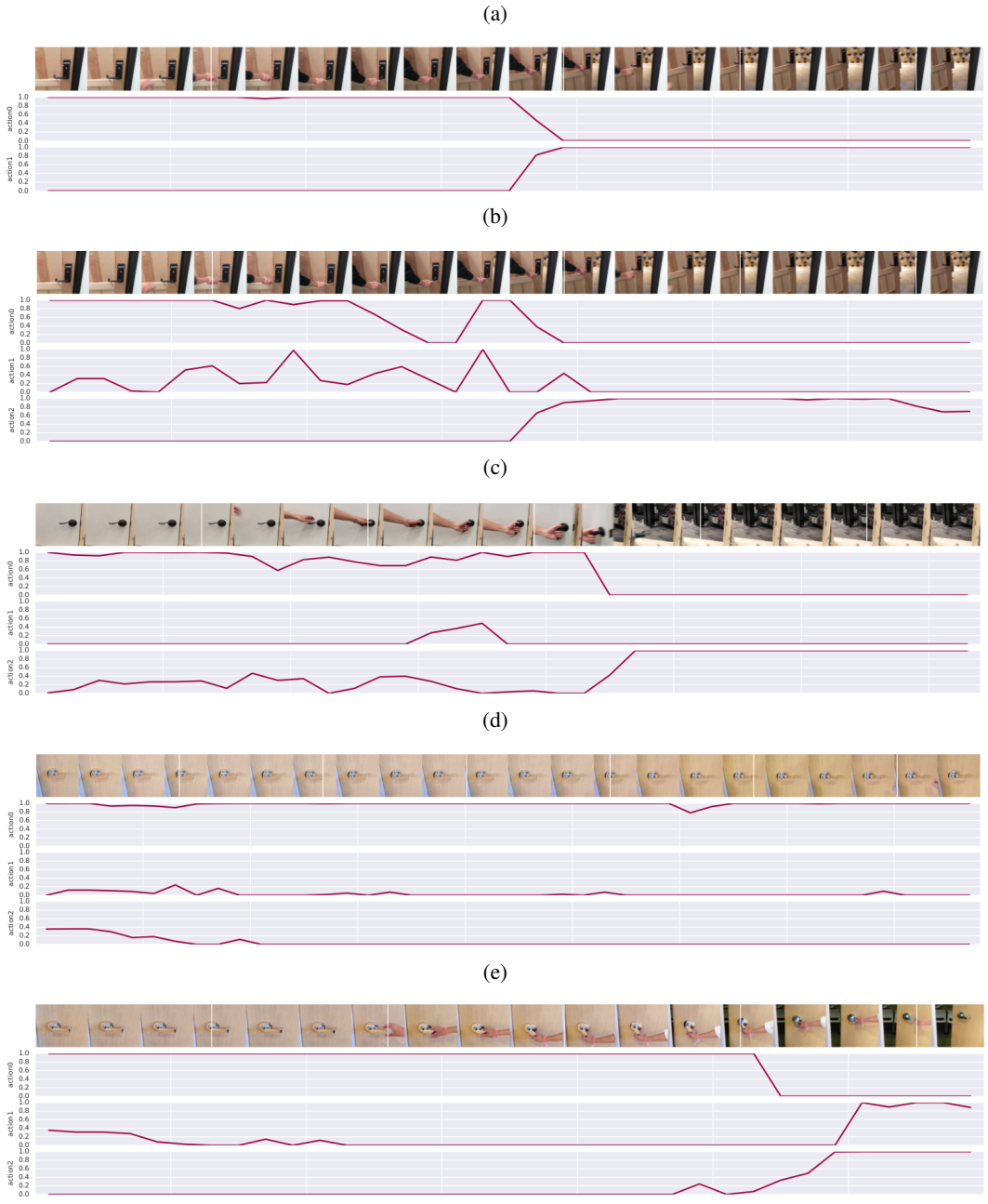


Figure 9: **Qualitative examples of reward functions for the door task** in testing videos. These plots show the individual sub-goal rewards for either 2 or 3 goals splits. The ”open” or ”closed” door reward functions are firing quite reliably in all plots, the ”hand on handle” step however can be a weaker and noisier signal as seen in 9b and 9c, or incorrect as shown in 9e. 9d demonstrates how a ”missed” action is correctly recognized.

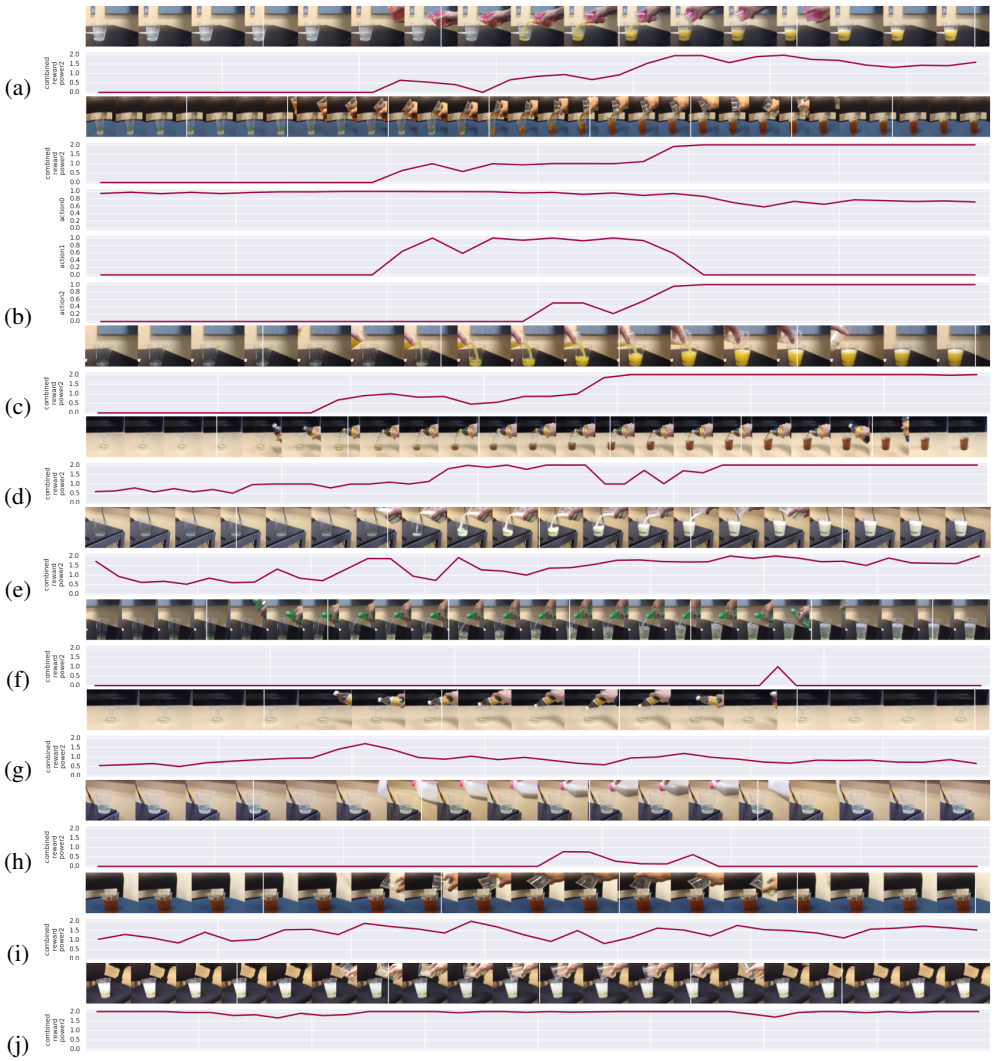


Figure 10: **Entire testing set of "pouring" reward functions.** This testing set is designed to be more challenging than the training set by including ambiguous cases such as pouring into an already full glass (10i and 10j) or pouring with a closed bottle (10g and 10h). Despite the ambiguous inputs, the reward functions do produce reasonably low or high reward based on how full the glass is. 10a, 10b, 10b and 10d are not strictly monotonically increasing but do overall demonstrate a reasonable progression as the pouring is executed to a saturated maximum reward when the glass is full. 10e also correctly trends upwards but starts with a high reward with an empty glass. 10f is a failure case where the somewhat transparent liquid is not detected.


Figure 11: **Entire training set of human demonstrations.**