A Gambler's DAgger: Reducing Supervisor Burden in Online Learning From Demonstration Learning Through Hierarchical Supervisors

Michael Laskey[1], Jonathan Lee[1], Caleb Chuck[1], David Gealy[1],...,Florian T. Pokorny, Anca D. Dragan[1], and Ken Goldberg[1,2]

***Abstract*— Online learning from demonstration algorithms such as DAgger can learn policies for problems where the system dynamics and the cost function are unknown. However they impose a burden on supervisors to respond to queries each time the robot encounters new states while executing its current best policy. Traditionally there has only been one supervisor and it is deemed expert. However, an expert supervisor might not be needed for a large portion of the state space, which could be handled by less-skilled supervisors. We present the Gambler's DAgger, which instead of only using an expert supervisors presents a hierarchical supervisor model, which first leverages cheaper supervisors, such as analytical methods on approximate dynamics or crowdsourced Amazon Mechanical Turk workers to train the policy and then only consults an expert for harder to learn states. We demonstrate this approach in training a visuomotor deep network policy for 2D grasping in clutter on a 4-DOF Zymark Robot. We were able to reduce the number of queries to the final expert supervisor by X and manage to collect XK examples by leveraging the hierarchical supervisor model.**

I. Introduction

In model-free robot Learning from Demonstration (LfD), a robot learns to perform a task, such as driving or grasping an object in a cluttered environment, from examples provided by a supervisor, usually a human. In such problems, the robot does not have access to either the cost function that it should optimize, nor the dynamics model. The former when it is difficult to specify how to trade-off various aspects that matter, like a car trying to drive on the road with other cars [4]. The latter occurs when either the system or the interaction with the world is difficult to characterize, like when a robot is trying to grasp an object in clutter and does not have an accurate model of the object dynamics.

Rather than explicitly learning the cost function as in Inverse Reinforcement Learning and the dynamics model, and then using optimization to produce a policy for the task, in model-free LfD the robot learns a policy directly from supervisor examples, mapping states to controls [5]. Learning from demonstration has been used successfully in recent year for a large number of robotic tasks, including helicopter maneuvering [2], car parking [3], and robot surgery [19].

LfD algorithms can be categorized as offline, where the robot only observes demonstrations, or online, where the robot interacts with the world and receives feedback from the supervisor. In offline LfD, the robot learns the policy based on a batch of examples, and then executes it to achieve the task. During execution, a small control error can accumulate, leading the robot away from the region of the state space where it was given examples, leading to unrecoverable failures. For example, a robot driving may be trained on examples driving safely down the center of a lane, but even slight deviations will eventually put the robot into states near the side of the road where its policy could fail [14]. Ross and Bagnell showed the number of errors made by the robot, in the worst case, can scale quadratically with the time horizon of the task [15].

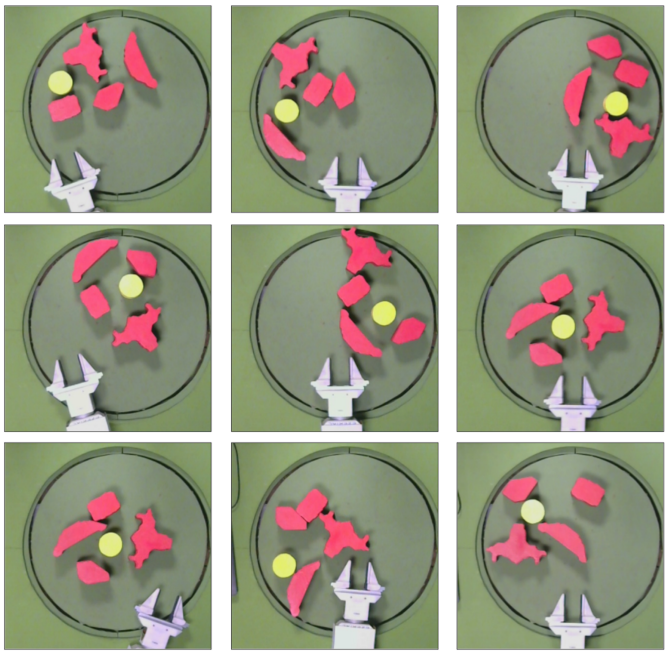


Fig. 1: Typical Initial States for our grasping in clutter task with a Zymark Robot. Red shapes indicate the clutter objects and the yellow circle indicates the goal object. The robot learns from raw image date to push the shapes and reach the yellow circle through our algorithm A Gambler's DAgger. The inscribed circle in the workspace prevents the robot from learning the trivial solution of simply pushing the red objects off the table.

Online LfD addresses this issue by iteratively requesting more examples from the supervisor in states the robot encounters [8], [15], [16]. One such algorithm, DAgger, learns a series of policies. At each iteration, the robot computes a policy based on the existing examples, then rolls out (executes) that policy, and the supervisor provides demonstrations for all states the robot visits. The new state/control examples are aggregated with the prior examples for the next iteration. DAgger and related algorithms have been applied in a wide range of applications, from quadrotor flight to natural language to Atari games [9], [7], [17]. In DAgger, under

[1] Department of Electrical Engineering and Computer Sciences; {mdlaskey,iamwesleyhsieh,ftpokorny,anca}@berkeley.edu, staszass@rose-hulman.edu

[2] Department of Industrial Engineering and Operations Research; goldberg@berkeley.edu

[1−2] University of California, Berkeley; Berkeley, CA 94720, USA

certain conditions, the number of errors scales only linearly with the time horizon of the task [16].

One drawback is that DAgger significantly increases the burden on the supervisor, who must label all states that the robot visits during training. Traditionally there has only been one supervisor and it is deemed an expert [15], [16], [17], [7]. However, an expert supervisor might not be needed for a large portion of the state space, which could be handled by less-skilled supervisors.

Our key insight is that for a large portion of the task the supervisor's time is used teaching the robot something that a less costly supervisor could do. For instance in our grasping in clutter example, the robot spends a large portion of the time trying to learn how to go towards the goal object. Analytical techniques such as forward kinematics, template matching and motion planning are readily available to provide examples in these parts of the state space. However, traditional Online Lfd method exploit the expert supervisor for all parts of the state space.

We propose bootstrapping a policy from a hierarchy of supervisors ranked by cost. Examples of such a hierarchy are analytical models with a simplified dynamics of the world, crowd-source Amazon Mechanical Turk workers and then finally an expert supervisor. We demonstrate this approach in training a deep neural network for the visuo-motor based task of 2D grasping in clutter on a Zymark Robot. Our robot consists of an 3DOF arm that lies in a planar workspace and the ability to rotate the table. The objects dataset consists of 20 Medium Density Fiber extruded polygons taken from 2D projections of objects in the Dex-Net [] database. Results show that by bootstrapping from less skilled experts, the total amount of burden on the expert supervisor is reduced and the task is successfully learned **ML: We need an objective measure of success**.

II. Related Work

Below we summarize related work in reducing the number of queries to an expert in the Online LfD setting.**ML: Still a work in progress**

Online Lfd with an Expert Supervisor Successful robotic examples of Online Learning From Demonstration where an expert have been employed have been in flying a quad-copter through a foster, navigating a wheel chair across a room and teaching a robot to follow verbal instructions and surgical needle insertion [17], [11], [7], [**?**].

However, to date these approaches have all used only an expert supervisor to label all parts of the state space. Our contribution consists of the use of a hierarchical structure of non-expert supervisors to reduce queries to the expert at the top of the hierarchy.

Reducing Supervisor Burden in Online LfD One approach that has been studied to reducing supervisor burden is applying active learning to only ask for queries one the robot is uncertain about the correct control to apply. Traditional active learning techniques like query-by-committee and uncertainty sampling have been applied by [6], [10], [8]

Kim et al. demonstrated that due to the non-stationarity of the problem traditional active learning techniques do not work because the underlying state distribution changes and thus the use of novelty detection is needed to be more conservative in determining uncertainty [11].

Laskey et al. created an approach, known as SHIV, for using active learning tailored to high dimensional and non-stationarity state distributions and provided a modified version One Class SVM to reduce the problem to regularized binary classification [**?**]. However, the result that novelty detection is needed to reduce supervisor burden via active learning implies that for problems with high stochasticity, such as grasping in clutter with a large number of objects, the expert would be queried a significant number. Furthermore, with the advent of deep learning where the feature space changes each iteration, it becomes unclear as to what feature space to provide novelty detection in.

Training Policies Via Analytical Methods Instead of relying on a costly human supervisor an alternative approach is to use knowledge of the underlying dynamics and train a policy via a computationally expensive analytical method such as a controller in a physics simulator [**?**] or dynamic programming in a Atari video game [9].

While this approach can "compile" the computationally expensive option into an quickly computed function, it can be limited when analytical models cannot explicitly model the environment. For instance in our grasping in clutter example, the robot has no knowledge of the mass or friction properties of the object, which would make this a challenging problem for such analytical methods. We thus propose using analytical methods not as the expert supervisor, but instead part of a hierarchy where they can be used to reduce the burden of the supervisors above them in the hierarchy.

III. Problem Statement

The goal of this work is to learn a policy that matches that of an expert supervisor's while asking expert supervisor with fewer queries than DAgger.

Modeling Choices and Assumptions We model the system dynamics as Markovian, stochastic, and stationary. Stationary dynamics occur when, given a state and a control, the probability of the next state does not change over time. Note this is different from the non-stationary distribution over the states the robot encounters during learning.

We model the initial state as sampled from a distribution over the state space. We assume a known state space and set of controls. We also assume access to a robot or simulator, such that we can sample from the state sequences induced by a sequence of controls. Lastly, we assume access to a set of supervisors who can, given a state, provide a control signal label. We additionally assume the supervisors can be noisy and imperfect, noting that the robot cannot surpass the level of performance of the current supervisor it is training with.

Policies and State Densities. Following conventions from control theory, we denote by $\mathcal{X}$ the set consisting of observable states for a robot task, consisting, for example, of high-dimensional vectors corresponding to images from a camera, or robot joint angles and object poses in the environment. We furthermore consider a set $\mathcal{U}$ of allowable control inputs for the robot, which can be discrete or continuous. We model dynamics as Markovian, such that the probability of state $\mathbf{x_{t+1}} \in \mathcal{X}$ can be determined from the previous state $\mathbf{x}_t \in \mathcal{X}$

and control input $\mathbf{u}_t \in \mathcal{U}$:

$$p(\mathbf{x}_{t+1}|\mathbf{u}_t, \mathbf{x}_t, \ldots, \mathbf{u}_0, \mathbf{x}_0) = p(\mathbf{x}_{t+1}|\mathbf{u}_t, \mathbf{x}_t)$$

We assume a probability density over initial states $p(\mathbf{x}_0)$ that can be sampled from.

A trajectory $\hat{\tau}$ is a finite series of $T+1$ pairs of states visited and corresponding control inputs at these states, $\hat{\tau} = (\mathbf{x}_0, \mathbf{u}_0, ..., \mathbf{x}_T, \mathbf{u}_T)$, where $\mathbf{x}_t \in \mathcal{X}$ and $\mathbf{u}_t \in \mathcal{U}$ for $t \in \{0, \ldots, T\}$ and some $T \in \mathbb{N}$. For a given trajectory $\hat{\tau}$ as above, we denote by τ the corresponding trajectory in state space, $\tau = (\mathbf{x}_0, ..., \mathbf{x}_T)$.

A policy is a function $\pi : \mathcal{X} \to \mathcal{U}$ from states to control inputs. We consider a space of policies $\pi_\theta : \mathcal{X} \to \mathcal{U}$ parameterized by some $\theta \in \mathbb{R}^d$. Any such policy π_θ in an environment with probabilistic initial state density and Markovian dynamics induces a density on trajectories. Let $p(\mathbf{x}_t|\theta)$ denote the value of the density of states visited at time t if the robot follows the policy π_θ from time 0 to time $t-1$. Following [16], we can compute the average density on states for any timepoint by $p(\mathbf{x}|\theta) = \frac{1}{T}\sum_{t=1}^{T} p(\mathbf{x}_t|\theta)$.

While we do not assume knowledge of the distributions corresponding to: $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$, $p(\mathbf{x}_0)$, $p(\mathbf{x}_t|\theta)$ or $p(\mathbf{x}|\theta)$, we assume that we have a stochastic real robot or a simulator such that for any state $\mathbf{x}_t$ and control $\mathbf{u}_t$, we can sample the $\mathbf{x}_{t+1}$ from the density $p(\mathbf{x}_{t+1}|\pi_\theta(\mathbf{x}_t), \mathbf{x}_t)$. Therefore, when 'rolling out' trajectories under a policy π_θ, we utilize the robot or a simulator to sample the resulting stochastic trajectories rather than estimating $p(\mathbf{x}|\theta)$ itself.

Objective. The objective of policy learning is to find a policy that maximizes some known cumulative reward function $\sum_{t=1}^{T} r(\mathbf{x}_t, \mathbf{u}_t)$ of a trajectory $\hat{\tau}$. The reward $r : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is typically user defined and task specific. For example, in the task of inserting a peg into a hole, a function on distance between the peg's current and desired final state is used [13].

While a reward function exists for a task like grasping in clutter, such as a positive value for successfully reaching the goal object. It is often too delayed to provide useful information for quickly learning [12]. We thus assume access to a supervisor that can achieve a desired level of performance on the task. The supervisor provides the robot an initial set of N stochastic demonstration trajectories $\{\tilde{\tau}^1, ..., \tilde{\tau}^N\}$. which are the result of the supervisor applying their policy. This induces a training data set $\mathcal{D}$ of all state-control input pairs from the demonstrated trajectories.

We define a 'surrogate' loss function as in [16], $l : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$, which provides a distance measure between any pair of control values. In the continuous case, we consider $l(\mathbf{u}_0, \mathbf{u}_1) = ||\mathbf{u}_0 - \mathbf{u}_1||_2^2$, while in the discrete case $l(\mathbf{u}_0, \mathbf{u}_1) = 1$ if $\mathbf{u}_0 \neq \mathbf{u}_1$ and $l(\mathbf{u}_0, \mathbf{u}_1) = 0$ otherwise.

Given a candidate policy π_θ, we then use the surrogate loss function to approximately measure how 'close' the policy's returned control input $\pi_\theta(\mathbf{x}) \in \mathcal{U}$ at a given state $\mathbf{x} \in \mathcal{X}$ is to the supervisor's policy's control output $\tilde{\pi}(\mathbf{x}) \in \mathcal{U}$. The goal is to produce a policy that minimizes the surrogate loss relative to the supervisor's policy.

Following [16], our objective is to find a policy π_θ minimizing the expected surrogate loss, where the expectation is taken over the distribution of states induced by the policy across any time point in the horizon:

$$\min_\theta E_{p(\mathbf{x}|\theta)}[l(\pi_\theta(\mathbf{x}), \tilde{\pi}(\mathbf{x}))] \tag{1}$$

If the robot could learn the policy perfectly, this state density would match the one encountered in user examples. But if the robot makes an error, that error changes the distribution of states that the robot will visit, which can lead to states that are far away from any examples and difficult to generalize to [14]. This motivates iterative algorithms like DAgger, which iterate between learning a policy and the supervisor providing feedback. The feedback is in the form of control signals on states sampled from the robot's new distribution of states.

IV. A Gambler's DAgger

In this section, we first recall the original DAgger algorithm. Then introduce the notion and formal definition of a hierarchical supervisor and finally describe how to combine DAgger with a hierarchical supervisor to create a Gambler's DAgger.

A. DAgger: Dataset Aggregation

DAgger [16] solves the minimization in Eq. 1 by iterating two steps: 1) compute a θ using the training data $\mathcal{D}$ thus far, and 2) execute the policy induced by the current θ, and ask for labels for the encountered states.

1) Step 1: The first step of any iteration k is to compute a θ_k that minimizes surrogate loss on the current dataset $\mathcal{D}_k = \{(x_i, u_i)|i \in \{1, \ldots, M\}\}$ of demonstrated state-control pairs (initially just the set $\mathcal{D}$ of initial trajectory demonstrations):

$$\theta_k = \arg\min_\theta \sum_{i=1}^{M} l(\pi_\theta(\mathbf{x}_i), \mathbf{u}_i). \tag{2}$$

This sub-problem is a supervised learning problem, solvable by estimators like a support vector machine or a neural net. Performance can vary though with the selection of a the estimator Selecting the correct function class depends on the task being consider and knowledge of the problem, see for a guide [18].

2) Step 2: The second step DAgger rolls out their policies, π_{θ_k}, to sample states that are likely under $p(\mathbf{x}|\theta_k)$. For every state visited, DAgger requests the supervisor to provide the appropriate control/label. Formally, for a given sampled trajectory $\hat{\tau} = (\mathbf{x}_0, \mathbf{u}_0, ..., \mathbf{x}_T, \mathbf{u}_T)$, the supervisor provides labels $\tilde{\mathbf{u}}_t$, where $\tilde{\mathbf{u}}_t \sim \tilde{\pi}(\mathbf{x}_t) + \epsilon$, where ϵ is a small zero mean noise term, for $t \in \{0, \ldots, T\}$. The states and labeled controls are then aggregated into the next data set of demonstrations $\mathcal{D}_{k+1}$:

$$D_{k+1} = \mathcal{D}_k \cup \{(\mathbf{x}_t, \tilde{\mathbf{u}}_t) \| t \in \{0, \ldots, T\}\}$$

Steps 1 and 2 are repeated for K iterations or until the robot has achieved sufficient performance on the task[1].

[1]In the original DAgger the policy rolled out was stochastically mixed with the supervisor, thus with probability β it would either take the supervisor's action or the robots. The use of this stochastically mix policy was for theoretical analysis. In practice, it is recommended to set $\beta = 0$ to avoid biasing the sampling [9], [16]

B. Hierarchy of Supervisors

Instead of one expert supervisor,we propose a hierarchy of M supervisors where for each supervisor $\tilde{\pi}_m$, there is an associated expected cumulative reward $R_m = \int \sum_{t=1}^{T} r(\mathbf{x}_t, \mathbf{u}_t) p(\mathbf{x}|\tilde{\pi}_m) d\mathbf{x})$, where $p(\mathbf{x}|\tilde{\pi}_m)$ denotes the average distribution of states the supervisor encounters. We also denote some measure of cost C_m that is ascribed to the supervisor, such as computational time or monetary expense. We assume that the rank of supervisors in terms of cost is equivalent to the rank of supervisor in the terms of expected cumulative reward. Thus, the cheapest supervisor also receives the lowest expected cumulative reward.

Additionally for two supervisors $\tilde{\pi}_m$ and $\tilde{\pi}_{m+1}$ to be in the hierarchy, they must only disagree in expectation on a subset of the work space to some precision $\tilde{\epsilon}$. Denote the set of states two supervisors disagree as $\mathcal{X}_{m,m+1} = \{\mathbf{x}|||\tilde{\pi}_m(\mathbf{x}) - \tilde{\pi}_{m+1}(\mathbf{x})||_2^2 > \tilde{\epsilon}\}$ .We make this condition because if two supervisors never provided the same examples then all states would have to be relabeled by the next supervisor in the hierarchy.

We formally define two supervisor's in a hierarchy as follows

Definition Two supervisors $\tilde{\pi}_m$ and $\tilde{\pi}_{m+1}$ are in a hierarchy if $C_m < C_{m+1}$, $R_m < R_{m+1}$ and $\mathcal{X}_{m,m+1} \subset \mathcal{X}$

We note that a two hierarchical supervisors according to this definition also implies that supervisor $\tilde{\pi}_{m+1}$ in the hierarchy is higher than the set of supervisors $\{\tilde{\pi}_i|\forall i \in [0, m]\}$.

C. A Gambler's DAgger

Our algorithm is as follows: first iterate through Step 1 and 2 of DAgger for a given supervisor, however after iteration K or when the current policy, $\pi_{\theta_{m,k}}$ is able to achieve a loss ϵ on the sampled J states at that current iteration (i.e):

$$\frac{1}{J}\sum_{j=1}^{J} ||\pi_{\theta_{m,k}}(\mathbf{x}_j) - \tilde{\pi}_m(\mathbf{x}_j)||_2^2 \leq \epsilon$$

Then iterate to the next supervisor in the hierarchy or $m = m + 1$. An issue arises though when performing this iterations because now the current dataset $\mathcal{D}_K$ has examples from a different supervisor that receives a smaller expected cumulative reward. This could cause a learning algorithm to try and regress to both labels and diverge from the intended performance [18].

We are thus interested in not storing the dataset collected from the previous supervisor and only remembering the current $\theta_{m,K}$ weight parameters. Thus, we propose only performing DAgger for each individual supervisor in the hierarchy and using the resulting weight vector to bootstrap the learning of the next supervisor. **ML: the last step about the dataset is something I want to empirically validate**

V. System for Grasping in Clutter Task

We now detail or experimental system for testing A Gambler's DAgger. First we describe the task itself, then each supervisor in the hierarchy and finally our deep network representation of the policy trained.

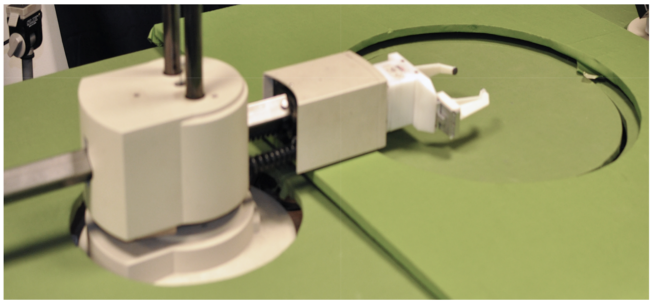


Fig. 3: Shown above is a Zymark robot. The robot consists of an 3DOF arm that lies in a planar workspace and the ability to rotate the turn table. The inscribed circle in the work space prevents the robot from learning the trivial policy of just pushing the objects off the table.

A. Task

We are interested in training a Zymark **ML: need make and model** to perform a grasping in clutter task on image data taken from a Logitech 2000 camera. Examples of images from the camera can be seen in Fig. 1.

The objects in clutter are made Medium Density Fiberboard and each one is on average 4" in diameter. The objects deemed cluttered are painted Red and the goal object is painted yellow. There is an inscribed circle around the work space to keep the robot from pushing the objects outside of the work space. The inscribed circle makes the task more challenging because the robot cannot simply "sweep" the cluttered objects off the table.

The robot, shown in Fig. 3, has a 4 dimensional internal state of base rotation, arm extension, gripper and turn table. The robot is commanded via state position requests, which are tracked by a tuned PID controller. The policy π_θ outputs delta positions that are bounded by $15\,\text{deg}$ for the gripper and turntable, 1 cm for the arm extension and 0.5 cm for the gripper at each time step. There is $T = 100$ time steps in a trajectory.

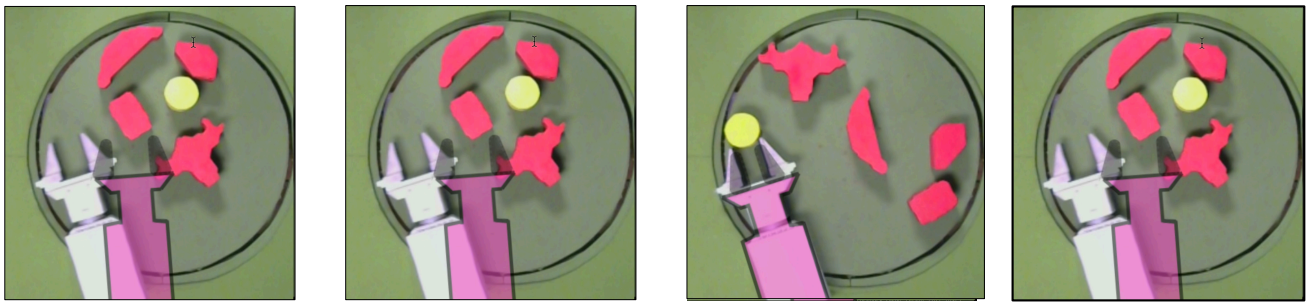
B. Supervisor Hierarchy

In this section, we list each supervisor in the order they appear in our hierarchy. The first supervisor is deemed the least expensive.

Analytical Supervisor Our first supervisor is a an analytical method that computes the trajectory the robot should move in to reach the goal object, or yellow circle. Our method employs template matching to identify the goal object in the image and the using the forward dynamics of the robot to computes the relative change in direction the robot should apply as an control. The template matching is implemented in OpenCV and uses the normalized cross-correlation filter [**?**].

In this stage, the analytical supervisors only tries to teach the robot to move towards the goal. Note that the analytical supervisor's advice contains no information about how the cluttered objects will respond to the dynamics applied, which results in sub-optimal polices. However, our analytical supervisor is both computationally and monetarily inexpensive to run which allows us to provide a large number of examples.

CrowdSourced Supervisor Our next level of supervisor relies on a crowd source service, called Amazon Mechanical Turk (AMT). Chung et al. demonstrated the reliable data learning from demonstration data could be obtained from an



(a) Rotation (b) Extension (c) Gripper (d) Turn Table

Fig. 2: The interface AMT workers see for providing feedback to the robot for the grasping in clutter task. The pink overlay indicates the desire change in the robot position with respect to the current robot state. Then AMT workers can use their intuition for how objects respond to force to provide examples of how the robot should behave. Each window shows a correction in a degree of freedom a) is the rotation of the robots base b) is extending the robot's arm c) is closing the gripper and d) is rotating the turn table

AMT platform. The AMT platform makes readily available thousands of human workers that can perform remedial task for a price range of $1-0.1[**?**]. Thus potentially providing a higher quality supervisor who has an intuition for how the cluttered objects interact with the world, but at a higher cost.

In order to get quality example from a CrowdSourced Supervisor, we designed an interface shown in Fig. 2. The interface draws a transparent overlay that shows how the robot would respond provided the current control this allows for the AMT worker to see some of the effect of their control. We rely on their natural intuition for how objects to infer how the change in robot position will change the objects.

We also design a tutorial for the work that first has them perform designated motions on a virtual robot and introduces them to a problem. We additionally provide a video of an expert providing corrections on three robot trajectories.

Expert Supervisor Finally after training with the Analytical and Crowdsourced Supervisor we leverage the Expert supervisor, who is capable of achieving high cumulative reward but is a limited resource, to provide corrections for a problem that would be non-obvious for someone who doesn't understand the physical limiations of the problem. Examples of this could be joint limits of the robot or how some of the objects might respond in more advanced dynamics. **ML: After we run the AMT study it will become apparent where an expert is still needed**

It can still be hard for an expert to provide the correct feedback without knowing the magnitude of the controls suggested, so they use the same interface as the CrowdSource Supervisor.

C. Deep Neural Network Policy Architecture

Our policy is represented as deep neural network, which was trained using TensorFlow [1]. Our network architecture consists of 1 convolutional layer with 5 channels, a fully connected layer with an output of 128 dimensions and a final layer that maps to a four dimensional control signal. We used ReLus to separate the different layers and a final tanh on the output to scale the output between -1 and 1.

To determine our architecture we created a set of 32,400 different architectures and then performed simulated annealing on the loss function trained after 400 iterations with a batch size of 200. The set of architectures consist of different network architectures as well as different momentum terms, and weight initialization schemes.

We trained all networks on a dataset of 6K images labeled with the Analytical Supervisor on a Nvidia Tesla K40 GPU, which is able to train each network in an average of 10 minutes.

VI. Experiments

ML: We need to nail down exactly what experiments are important. I suggest 1) main one is to show expert time withouth bootstrapping vs. with 2) can be different forms of how to handle the dataset aggregation (i.e. full retrain or batch update etc.) 3) analysis of the crowd source system and how effective that was (this would be a secondary follow up experiment though, 4) Can be also comparison with a previous way to reduce supervisor burden in high dimensional states (i.e. SHIV).

VII. Discussions and Future Work

VIII. Acknowledgments

This research was performed in UC Berkeley's Automation Sciences Lab under the UC Berkeley Center for Information Technology in the Interest of Society (CITRIS) "People and Robots" Initiative. This work is supported in part by the U.S. National Science Foundation under Award IIS-1227536, NSF-Graduate Research Fellowship, by the Knut and Alice Wallenberg Foundation and the National Defense Science and Engineering Graduate Fellowship Program. We thank Pieter Abbeel, Sergey Levine and Sanjay Krishnan for insightful feedback.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[2] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," *NIPS*, vol. 19, p. 1, 2007.

[3] P. Abbeel, D. Dolgov, A. Y. Ng, and S. Thrun, "Apprenticeship learning for motion planning with application to parking lot navigation," in *IROS 2008. IEEE/RS.* IEEE.

[4] P. Abbeel and A. Y. Ng, "Apprenticeship learning via reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 1.

[5] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[6] S. Chernova and M. Veloso, "Interactive policy learning through confidence-based autonomy," *Journal of Artificial Intelligence Research*, vol. 34, no. 1, p. 1, 2009.

[7] F. Duvallet, T. Kollar, and A. Stentz, "Imitation learning for natural language direction following through unknown environments," in *ICRA*. IEEE, 2013, pp. 1047–1053.

[8] D. H. Grollman and O. C. Jenkins, "Dogged learning for robots," in *ICRA, 2007 IEEE*.

[9] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang, "Deep learning for real-time atari game play using offline monte-carlo tree search planning," in *NIPS*, 2014, pp. 3338–3346.

[10] K. Judah, A. Fern, and T. Dietterich, "Active imitation learning via state queries," in *Proceedings of the ICML Workshop on Combining Learning Strategies to Reduce Label Cost*, 2011.

[11] B. Kim and J. Pineau, "Maximum mean discrepancy imitation learning." in *Robotics Science and Systems*, 2013.

[12] N. Kitaev, I. Mordatch, S. Patil, and P. Abbeel, "Physics-based trajectory optimization for grasping in cluttered environments," pp. 3102–3109, 2015.

[13] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *arXiv preprint arXiv:1504.00702*, 2015.

[14] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," Carnegie-Mellon University, Tech. Rep., 1989.

[15] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 661–668.

[16] S. Ross, G. J. Gordon, and J. A. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," *arXiv preprint arXiv:1011.0686*, 2010.

[17] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, "Learning monocular reactive uav control in cluttered natural environments," in *ICRA, 2013 IEEE*. IEEE.

[18] B. Schölkopf and A. J. Smola, *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[19] J. Van Den Berg, S. Miller, D. Duckworth, H. Hu, A. Wan, X.-Y. Fu, K. Goldberg, and P. Abbeel, "Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations," in *ICRA, 2010 IEEE*. IEEE, 2010, pp. 2074–2081.