Robotic Grasping in Clutter: Using a Hierarchy of Supervisors with (Deep) Learning from Demonstrations

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

Abstract— **Recent progress in learning from demonstrations increasingly indicates that robots can acquire manipulation skills from large collections of training data. Online learning from demonstration algorithms such as DAgger can in particular learn policies for problems where the system dynamics and the cost function are unknown. Traditionally, these systems have been trained by skilled human supervisors providing feedback on the current learned policy. As the number of demonstrations increases, this approach can however incur a substantial time and labor cost. We propose that a hierarchy of supervisors that vary is expertise but also in cost can help alleviate this issue. We combine planning on a relaxed problem with crowdsourcing and with robotics experts, and hypothesize that leveraging this combination can lead to similar performance while substantially decreasing cost. We focus on the grasping in clutter problem, and use an architecture for learning from demonstration based on DAgger and deep learning to learn policies directly from image input. Our results ....[fill in]**

I. Introduction

As illustrated by the recent Amazon Picking Challenge at ICRA 2015, the grasping in clutter problem, where a robot needs to grasp an object that might be occluded by other objects in the environment poses an interesting challenge to robotic systems. Two fundamental approaches to grasping in clutter include the analytic model driven approach [?], [?], [?], where the interaction dynamics between the robot and obstacles are formulated analytically. However, modeling all the physical properties of interaction poses a highly challenging problem due to uncertainty in modeling parameters such as inertial properties and friction.

Another approach to the grasping in clutter problem in a data-driven manner, where the interaction behavior is learned directly from interactions with the environment and a supervisor, which can be an expert human or an analytical method [4]. Learning from demonstration (LfD) algorithms have been used successfully in recent years for a large number of robotic tasks, including helicopter maneuvering [2], car parking [3], and robot surgery [25]. Furthermore, deep learning, which we use to learn policies directly from raw video data, has emerged as a highly versatile technique used for this purpose [19].

Our approach is based on online Learning from Demonstrations (LfD) where a robot iteratively learns a policy and is provided by feedback on the current policy roll out by a supervisor [9], [20], [21]. We build on, DAgger, an online LfD algorithm which at each iteration, computes a policy based on prior demonstrations, and then rolls out that policy. A supervisor then provides control signals as feedback on the current policy and 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 [10], [8], [22]. Ross et al. showed that DAgger, under a no-regret assumption, can be guaranteed to deviate from the supervisor's policy with an error at most linear in the time horizon for the task [21].

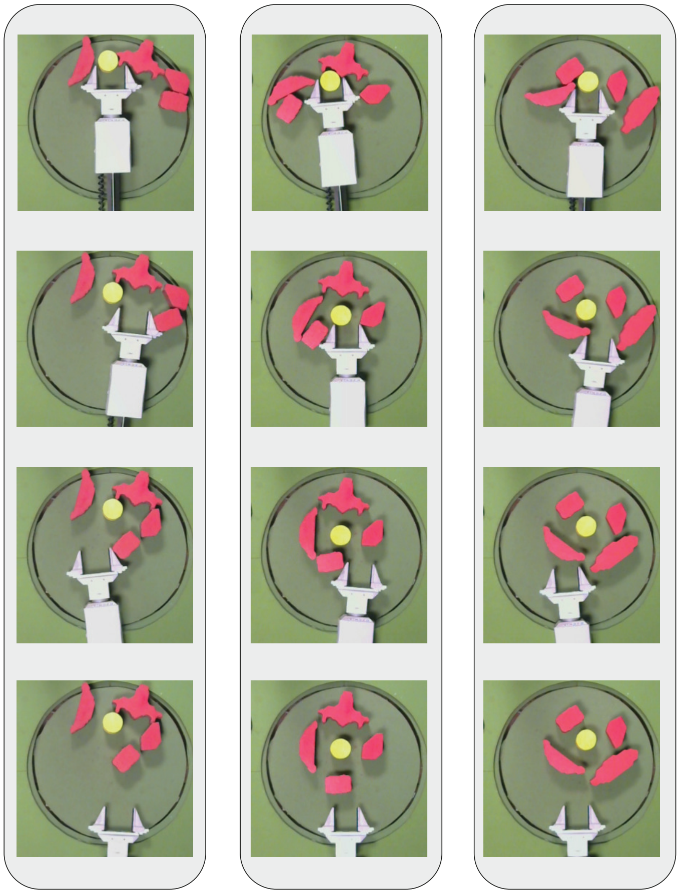


Fig. 1: Three roll-outs on a Zymark 3-DOF Robot of a fully trained grasping in clutter policy (one per column, bottom to top) which was trained using a hierarchy of three supervisors consisting of an analytical motion planning, crowd-sourcing and human expert. Red shapes indicate clutter objects and the robot is trained to reach the yellow circle. The trained manipulation policy is represented as a deep neural network that recieves as input an image of the scene and outputs a change in state position. The resulting policy learns to sweep away clutter objects and reach the goal object.

One drawback is that DAgger imposes a substantial burden on the supervisor, who must label all states that the robot

[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

visits during training. Traditionally there has only been one supervisor and it is deemed an expert [20], [21], [22], [8]. In this work, we propose to instead utilize a hierarchy of supervisors that exhibit increasing cost and competence at a given task to incrementally bootstrap the learning process.

In particular, while model-based planners might not yet capture the full physical interaction among all objects, it can be wasteful to not leverage them and learn with an expert supervisor from scratch. Thus, at the lowest lever of the hierarchy, we use a motion planner on an relaxed version of the problem. In our grasping application, we use a planner that ignores the obstacles but can leverage knowledge of the kinematics and the goal. The next supervisor in the hierarchy are crowdsource workers on Amazan Mechnical Turk. Then finally the robot is trained with a Phd student in robotics, who is an expert human supervisor. Examples of the resulting learned grasping in clutter policy can be seen in Fig. 1.

II. Related Work

Below, we summarize related work in Robotic Grasiping in Clutter, the Online LfD setting and then the field of Curriculum learning, which is related to the concept of hierarchical supervisors.**AD: make sure to talk about guided policy search too, which is a way to combine deep nets with either human or planner supervision**

Robotic Grasping in Clutter Robotic grasping is a well-established research topic in robotics that has been studied for several decades [6]. A significant amount of prior work focuses on planning grasps given a known object. However, in unstructured environments clutter poses a significant challenge for reaching the planned grasp [12]. Prior work has addressed integrated perception and grasping in clutter where the objective is to grasp objects in an unorganized pile [19], [18], but these methods do not specifically aim to grasp a single target object in clutter. Leeper et al. [] use a human operator for assistance to guide the robot through clutter with the objective of grasping a target object. However, the robot required the human to be present at all times and did not attempt to learn to operate autonomously. We are interested in a data-driven approach that only queries a supervisor at training time and which can operate autonomously thereafter.

Prior work has studied the problem of manipulating objects by performing pushing operations []. Berenson et al. [] in particular use a sampling-based planner that considers clearance from obstacles in the environment to plan a grasp approach trajectory in cluttered environments. Cosgun et al. [3] and King et al. [14] consider the problem of planning a series of push operations that move an object to a desired target location. Kiteav et al. planned a trajectory in a physics simulator using LQG based controllers [15]. However, all of these works assume a known model of of the object behavior is provided. We are interested in learning manipulation polices in a data-driven manner instead, leveraging a hierarchy of supervisors for training. **ML: We should discuss how to distinguish our work (Properties of our approach are we assume raw image data, real time and unknown dynamics)**.

Online Lfd with an Expert Supervisor Successful robotic examples of Online Learning From Demonstration with an expert supervisor include applications to flying a quad-copter through a forest, navigating a wheel chair across a room and teaching a robot to follow verbal instructions and surgical needle insertion [22], [13], [8], [**?**].

However, to date, these approaches have used only one expert supervisor to provide training data for all parts of the state space. We propose to instead utilize hierarchy of supervisor of different skill level and cost to reduce the overall learning cost.

Reducing Supervisor Burden in Online LfD One approach that has been studied to reduce the number of supervisor supervision is to apply active learning to only ask for supervision when the robot is uncertain about the correct control to apply. Traditional active learning techniques like query-by-committee and uncertainty sampling have in particular be utilized for this purpose [7], [11], [9]

However, Kim et al. demonstrated that due to the non-stationarity of the distribution of states encountered during learning, the traditional active learning techniques may be suitable since the underlying state distribution changes. Thus the use of novelty detection was proposed [13]. Laskey et al. introduced SHIV, using an active learning approach tailored to high dimensional and non-stationarity state distributions and a modified version of the One Class SVM classifier. This enabled the authors to reduce the density estimation problem to a simpler regularized binary classification [**?**]. However the grasping in clutter problem exhibits a high amount of stochasticity requiring a large training data set which poses a significant computational challenge to these methods. In the present paper, we hence consider using Deep Learning, as an underlying scalable learning algorithm in combination with DAgger.

Curriculum Learning Our approach is closely related to ideas from curriculum learning, where a neural network is trained via incrementally, first on easier examples and then gradually on data of increasing difficulty [5].

Sanger et al. used curriculum learning in robotics to gradually train a neural network policy to learn the inverse dynamics of a robot manipulator. They then considered a collection scheme where easily learned trajectories where shown to the robot first and then gradually increased the difficulty [**?**].

Our approach is different from curriculum learning, which assumes all training data presented to the neural network is valid. We instead train on a completely different set of examples throughout the supervisor hierarchy. In these different set of examples, some could be very low-quality examples and will enforce a behavior that needs to be retrain with a more-skilled supervisor.

III. Problem Statement

Given a collection of increasingly able and costly supervisors $S_1, \ldots, S_M$, the goal of this work is to learn a policy that closely matches that of the most able supervisor S_M while minimizing the overall cost of training a policy. We formalize this approach as follows.

Assumptions and Modeling Choices 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 a lower cost supervisor also has lower quality.

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.

Policies and State Densities. Following conventions from control theory, we denote by $\mathcal{X}$ the set 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 denote by $\mathcal{U}$ the set of allowable control inputs for the robot. $\mathcal{U}$ may be discrete or continuous in nature. We model dynamics as Markovian: the probability of state $\mathbf{x_{t+1}} \in \mathcal{X}$ depends only oni 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 an unknown probability density over initial states $p(\mathbf{x}_0)$.

A demonstration (or trajectory) $\hat{\tau}$ is a 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 [21], 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 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, 'rolling out' trajectories under a policy π_θ in our experiments, we utilize the robot to sample the resulting stochastic trajectories rather than estimating $p(\mathbf{x}|\theta)$ itself.

FP: might make this and the above specific to grasping in clutter? **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 quantifying a notion of distance between the peg's current and desired final state can be used [16].

Since grasp success is typically considered as a binary reward which is observed only at a delayed final state [15], grasping in clutter poses a challenging problem for traditional reinforcement learning methods. We hence instead build upon DAgger which queries a supervisor for appropiate actions, to provide the robot a set of N stochastic demonstrations trajectories $\{\hat{\tau}^1, ...\hat{\tau}^N\}$. This induces a training data set $\mathcal{D}$ of state-control input pairs. We define a 'surrogate' loss function as in [21], $l : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$, which provides a distance measure between any pair of control values. We consider $l(\mathbf{u}_0, \mathbf{u}_1) = ||\mathbf{u}_0 - \mathbf{u}_1||_2^2$.

Given a candidate policy π_θ, we DAgger uses the surrogate loss function to approximately measure how 'close' the robot's 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 of DAgger is to produce a policy that minimizes the expected surrogate loss:

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

Instead of a single supervisor S, which classically is considered to be a skilled human teacher, we instead consider a hierarchy $S_1, \ldots, S_M$ of supervisors which we may be algorithms or humans and which follow policies $\pi_1, \ldots, \pi_M$ with associated expected cumulative rewards R_i satisfying $R_1 \leq R_2 \leq \ldots \leq R_M$. Furthermore, we assume that the cost associated to providing a state label for supervisor S_i is C_i with $C_1 \leq C_2 \leq \ldots C_M$, so that the ordering of supervisors is consistent with respect to both cost and skill level. We consider the problem of minimizing the expected surrogate loss of a trained policy with respect to the most skilled supervisor S_M in the hierarchy while minimizing the overall training cost.

In particular, this paper provides an empirical study of greedy combinations of three types of supervisors S_1, S_2, S_3 for grasping in clutter which are applied in order to train a policy parameterized by a deep neural network. Here, S_1 is an analytical motion planning algorithm with a cost $C_1 = 0.01$, S_2 a supervisor consisting a crowd-sourced Amazon Mechanical Turk laborers with $C_2 = 0.1$, and S_3 a human expert supervisor with $C_3 = 1$.

IV. Approach and Background

A. Details on DAgger: Dataset Aggregation

Since the cumulative expected reward of a policy is difficult to optimize directly, DAgger [21] instead solves the minimization in Eq. 1 by iterating two steps: 1) computing the policy parameter 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):

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

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 [23].

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 $t \in \{0, \dots, T\}$. The states and labeled controls are then aggregated into the next data set of demonstrations $\mathcal{D}_{k+1}$:

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

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

B. DAgger with Supervisor Hierarchy

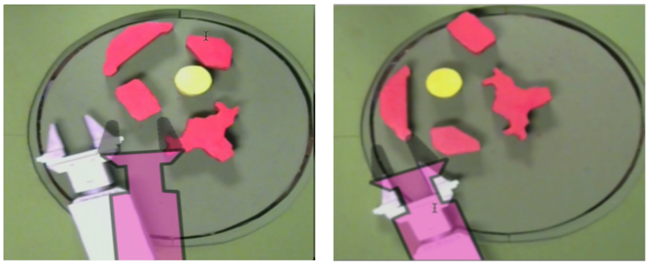
Formulating a framework for selecting a supervisor can be difficult because we do not have a known model for how a supervisor selected affects the minimization of the surrogate loss. The reason we do not have a model is because this would require modeling what examples the supervisor will return, the θ parameters of the neural network policy after training and the states the robot is likely to visit given those parameters.

We could learn a policy using model-free reinforcement learning algorithms [24], where a selection strategy will query different supervisors and learn over time the best supervisor to select given the current policy. However, a problem occurs when applying these approaches because it requires evaluating the objective function, which is the surrogate loss with respect to the costliest and most-skilled supervisor, S_M. This requires exhaustively querying the costliest supervisor at each iteration and defeating the purpose of reducing cost.

In light of this, we propose a greedy allocation strategy, in terms of supervisor cost, where we train a policy with the cheapest supervisor first for a fix number of trials and then advance through the hierarchy training with each supervisor. Our algorithm is as follows: first iterate through Step 1 and 2 of DAgger for a given supervisor, however after iteration K_m. Then iterate to the next supervisor in the hierarchy or $m = m + 1$.

The number of iterations before advancing to the next supervisor can be challenging to set. In practice, we either advanced when the surrogate loss between supervisors was sufficiently low or when the performance on the grasping in clutter task was not improving. The reason for different options was because some supervisors have very large variance (i.e. crowdsourcing), that a fix threshold on surrogate loss could be misleading without a sufficient number of examples.

An issue arises though when performing this iterations because now the current dataset $\mathcal{D}_{K_m}$ has examples from a different supervisor that receives a smaller expected cumulative reward. This could cause a learning algorithm to try and fit to contradictory examples and be worse than either supervisor trained with [23].



(a) Rotation (b) Extension

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.

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. We found empirically in Sec. VI that it also beneficial to add to the new dataset the states and controls applied by the current policy $\pi_{\theta_{m,k}}$ that the current supervisor agreed with by some euclidean distance $\tilde{\epsilon}$.Thus, acting as a regularization on the optimization since stochastic gradient on a section of the min-batch update would be zero for examples where the next supervisor agrees with the previous.

V. Robot Grasping in Clutter

We now describe the grasping in clutter problem, the supervisor hierarchy that we used to solve it and our deep learning policy architecture.

A. Grasping in Clutter

Consider a robot in an Amazon warehouse. The robot has an intended goal object for it to grasp on a shelf, however other objects could obstruct the path towards the goal object. These obstructions of movable objects prevent leveraging common motion planning techniques, because they do not posses a dynamics model of how the objects behave under different contact forces. [15], [14].

Thus, the robot must reason about how its interactions with the environment will affect the intended outcome. This can be hard for two reasons 1) it becomes hard to find a low dimensional state representation for the task because the environment dynamics are affected by the global shape of each object 2) the dynamics of the environment involve modeling how K objects interact with each other under an arbitrary force [15].

In order to achieve real time grasping in clutter, which is important for a robot in a shipping warehouse, we leverage visuo-motor deep learning to learn a control policy for the task over a large number of different configurations working with a hierarchy of supervisors. Deep learning has the potential to take high dimensional image data of the scene and learn task specific features relevant for grasping in clutter. Additionally a trained neural network policy is can be computational

[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 [10], [21]

inexpensive to evaluate and can result in real time performance [16].

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.

Motion Planning that Ignores Obstacles Supervisor Our first supervisor, S_1, is a a motion planner that that computes the trajectory the robot should move in to reach the goal object, or yellow circle, by ignoring obstacles. Thus we refer to S_1 as Motion Planning that Ignores Obstacle (MPIO) supervisor. 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 MPIO supervisors only tries to teach the robot to move towards the goal. Note that the MPIO supervisor's examples does not have any knowledge about how the cluttered objects will respond to the forces applied via the robot arm, which results in sub-optimal polices. However, the MPIO supervisor is both computationally and monetarily inexpensive to run which allows us to provide a large number of examples. Thus, the cost of the MPIO supervisor is $C_1 = 0.01$.

CrowdSourced Supervisor The second supervisor, S_2, relies on a crowd source service, called Amazon Mechanical Turk (AMT). The AMT platform makes readily available thousands of human workers that can perform provide examples for $0.12 per robot trial. 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, $C_2 = 0.1$.

In order to get examples 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 Thus, allowing for the AMT worker to see the magnitude and effect their example would have on the robot.

We additionally designed a tutorial for the CrowdSourced Supervisor. First, we introduce them to a robot named Izzy and briefly explain that Izzy is trying to reach a yellow circle but has a tendency to "misbehave" and needs their help. Then we have them perform designated motions with a virtual Izzy, which helps them understand how the robot's dynamics behave. We additionally provide a video of an expert providing corrections on three robot trajectories. We lastly instruct the CrowdSourced Supervisor to only label when they are sure in their decision, in order to prevent contradictory examples associated with them second guessing themselves.

While completing a tutorial before providing examples can be beneficial, we still are not sure whether the CrowdSourced Supervisor can provide quality examples that can be used in learning. To help ensure quality, we give all CrowdSourced Supervisor the same robot trial first and measure their examples against the examples form a Human Expert Supervisor. If the average Squared-Euclidean distance between examples is above a threshold of the Human Expert's examples or the number of examples they provided is above a threshold of the number the Human Expert provided, the Crowdsource Supervisor is not asked to provide additional examples.

Human Expert Supervisor The final supervisor, S_3 is the Human Expert supervisor, who is capable of achieving high cumulative reward but is a limited resource. An Human Expert Supervisor in this case is a Phd student in machine learning and robotics, which costs $C_3 = 1.0$. The Human Expert Supervisor uses the same interface , shown in Fig. 2, to provide examples to the robot.

An expert supervisor can be used in a variety of scenarios.They first would have a better intuition of the physical limitations of the robot and environment, such as joint limits or how certain objects might behave under force. Furthermore, they would also understand how the examples given could lead to better training of the robot's policy π_θ. For example, understanding the feature space the policy lies in can lead the Human Expert to not providing contradictory examples.

C. Neural Network Policy Architecture

ML: adding more details with figure tomorrow 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 and filters with size 11x11, 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.

The control examples was scaled between 1 and -1 for each dimension independently. To be robust to lighting and reduce the dimensionality of the problem, we applied a binary mask to each channel of the 250x250 RGB image, the mask would set to 1 values above 125 and 0 other wise. We then validated that all information (i.e. location of the gripper, cluttered shapes and goal object) where still visible in the masked image.

To determine our architecture we preformed a grid search over different architectures 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

In this section, we first describe the grasping in clutter experimental setup. Then we test how the supervisor hierarchy performs versus not leveraging a hierarchy. We then test how high of Quality the CrowdSource supervisor is. Then, we experiment with different ways to advance in the hierarchy. Finally, we look at using all three supervisors and to train a robot for the grasping in clutter task.

A. Experimental Setup

For experimenting in the grasping in clutter domain, we are interested in training a Zymark robot to perform a grasping in clutter task on image data taken from a Logitech C270

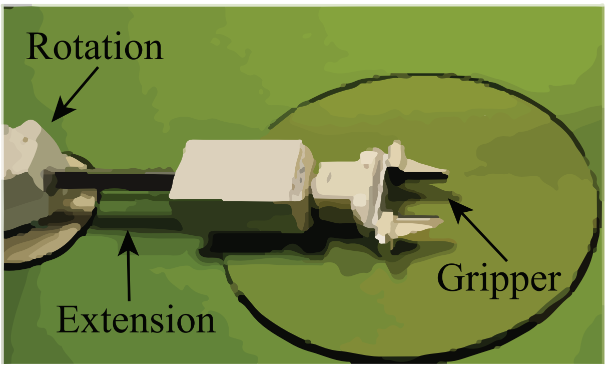


Fig. 3: Shown above is a Zymark robot. The robot consists of an 3DOF arm that lies in a planar workspace and the ability to rotate the turn table. The inscribed circle in the work space prevents the robot from learning the trivial policy of just pushing the objects off the table.

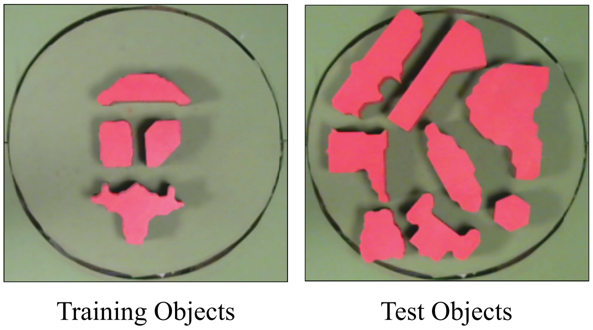


Fig. 4: The set of objects that the robot was trained on. The Training objects on the right are the four objects used in training. The test objects on the left represent represent objects that were found in our test configurations. The test objects vary in size and shape from the training objects, which can test how well the robot learns to manipulate unknown objects. Every test configuration contained at least one object from this set to guarantee it wasn't trained on.

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 can make the task more challenging because the robot cannot simply "sweep" the cluttered objects off the table.

The robot, shown in Fig. 3, has a 3 dimensional internal state of base rotation, arm extension and gripper. The robot is commanded via state position requests, which are tracked by a tuned PID controller. We used relative state contorl because 1) it is easier to enforce stay out zones (such as where a human operator could be) in the robot configuration space and 2) registering state control to a labeling interface in Fig. 2 is more straightforward than motor commands. The policy π_θ outputs delta positions that are bounded by 15° 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.

To test the performance of a policy, we created a test set composed of 20 different configurations each containing objects that were not trained on. The test set configurations varied in number of objects on the table, size of objects and relative pose of each object. The objects that were trained on and those that were only tested on can be seen in Fig. 4. We measure success as defined by three situations not successful, near successful, successful. Examples of the robot in these situations can be seen in Fig. 5.

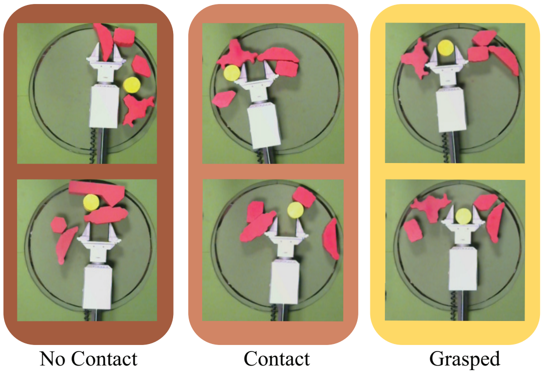


Fig. 5: The different measure of success used in our scoring metric for how well the robot performed. Not successful corresponds to when the gripper is never in contact with yellow circle. Near successful corresponds to when the robot ends with the gripper touching the yellow circle. Successful is defined as when the yellow circle is inside the gripper.

The not successful situation corresponds to the case when the robot's gripper is not touching the goal object at the end of a rollout. Examples of when this happens is when the robot fails to push the objects away or does not head in the direction of the yellow circle. The near successful situation is defined as the robot touches the goal object with its gripper, but it is not enclosed in the gripper. Examples of when this happens is when the robot gets a small obstacle object trapped in the gripper first, or when the root slightly misses the yellow circle on its approach. Successful is defined as getting the goal object inside the gripper at the end of a trajectory. We report total success as a score out of 2: 0 for not successful, 1 for near successful, 2 for successful.

In all experiments we used a batch size of 20 trials per iterations or 2000 images, thus 100 trials corresponds to 5 iterations.

B. Hierarchical Supervisors

AD: reorder: mpio, he, mpio-¿he (main experiment); then cs, mpio-cs, all; cs is missing score, hopefully it does not much better than mpio-cs; don't highlight bottom right option because technically its not suppod to be better than expert; just use hypotheses to show your points.; also separate mpio-cs-he from rest becuase it is using double the data! not fair comparison! spread the legend horizontally at the top;

We first test that using a hierarchy of supervisors can reduce total cost needed to train a policy, but still maintain similar performance . We compare having only a MPIO Supervisor for a fixed amount of data to having only an Human Expert, and to having the hierarchy of both . The non-hierarchical policies trained with the human expert supervisor only for 160 demonstrations or only the MPIO supervisor for 160 demonstrations. The policy trained with a hierarchical supervisor first receives 100 demonstrations from the MPIO supervisor and 60 demonstrations from the expert supervisor.

Our results, shown in Fig. 6, are the policy trained with just

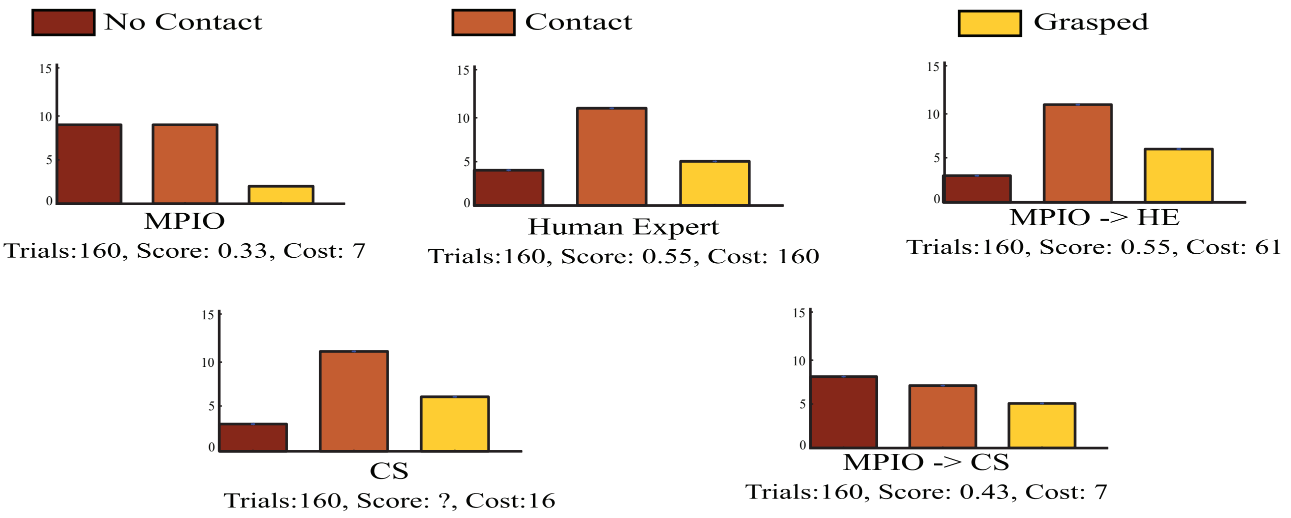


Fig. 6: The performance and cost of each policy trained with a supervisor is reported. The bar graphs shows the breakdown in terms of situations the policy encountered on the test set. The top row corresponds to policies only trained with one supervisor: MPIO, Crowdsourced or Human Expert. The bottom corresponds to policies trained with a hierarchy of supervisors: MPIO and Crowdsource, MPIO and Human Expert, MPIO, Crowdsourced and Human Expert. The final bottom right plot demonstrates the full hierarchy, which achieves the best performance (0.8).

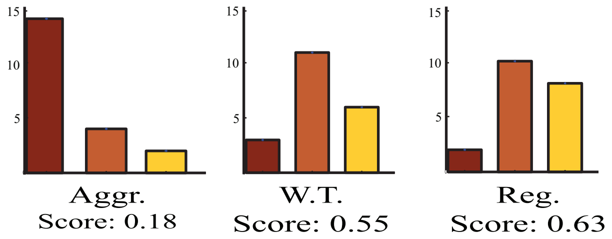


Fig. 7: The performance of each policy trained with a different data management strategy is reported. The bar graphs shows the breakdown in terms of situations the policy encountered on the test set. Each policy was trained with the MPIO to Human Expert hierarchical supervisor. From left to right is the following data management strategies: Dataset Aggregation, Weight Transfer and Regularization. Regularization achieves the highest performing score of (0.18)

the MPIO supervisor achieves a score of 0.325%, the policy trained with just an expert achieves 0.54% and the policy trained with a hierarchy of supervisor achieves 0.55%. Thus, the hierarchical supervisor and expert supervisor achieve approximately the same performance. However,training with a hierarchical supervisor yields a 40% reduction in cost incurred compared to the policy trained with only an expert supervisor.

Thus demonstrating that by using a hierarchy of supervisors, we can reduce the cost of training a policy and achieve similar performance to the Human Expert Supervisor.

C. Quality of Crowdsourced Supervisor

We next evaluate the potential of a crowdsourced supervisor for a being part of the hierarchy. Thus, we perform an experiment using the AMT platform described in Sec. V-B. We first test how well a crowdosurce supervisor performs as part of a hierarchy of supervisors. We trained a policy with 100 demonstrations from the MPIO supervisor , then had 60 demonstrations provided by AMT workers. We also compare how well the crowdsource supervisors performs just as well as a single supervisor, by having them provide all 160 demonstrations.

Our results, shown in Fig. 6, are the policy trained with just the crowdsourced supervisor achieves a score of $?\%$. The policy trained with the hierarchical crowdsourced and MPIO supervisor receives 0.43% on our test set compared to 0.55% with the motion planning and expert supervisors hierarchy. The total cost received with the only crowdsource supervisor was 15, while the hierarchical supervisor was 7.

In total, we tested on 44 AMT workers. We asked them to provide examples for a single rollout and then if they passed quality check asked them if they wanted to exit or continue for an additional \$0.12 per demonstration. We found only 21 AMT workers were able to pass the quality check and continue providing demonstrations. On average a worker would provide 4 demonstrations before exiting the AMT HIT.

D. Advancing in the Hierarchy

We further test how to advance between supervisors in the hierarchy.We examine three different strategies for dataset management when changing supervisors 1) aggregating the dataset of the data collected with both supervisors 2) transferring only the weights of the nerural network policy or θ vector between the supervisors and removing the first's supervisors dataset completely 3) transferring only the weights of the neural network policy but adding the values output by $\pi_\theta(\mathbf{x})$ instead of $\tilde{\pi}_m(\mathbf{x})$at parts of the states visited that the current supervisor is close to agreement with as measured by the following $||\tilde{\pi}_m(\mathbf{x}) - \pi_\theta(\mathbf{x})||_2^2 < 0.01$. Thus, acting as a regularization on the optimization since stochastic gradient on a section of the min-batch update would be zero for examples where the next supervisor agrees with the previous.

We compare each strategy on a hierarchy of a MPIO supervisor trained with 100 iterations and then advancing to a human expert supervisor trained with 60 iterations. Our results reported in Fig.7 , show that aggregation strategy achieves 0.18%, the weight transfer strategy achieves 0.55% and the regularized stochastic gradient update strategy achieves 0.63%. Thus, suggesting that techniques to help the policy remain close to the previously trained supervisor are useful for effectively switching between supervisors.

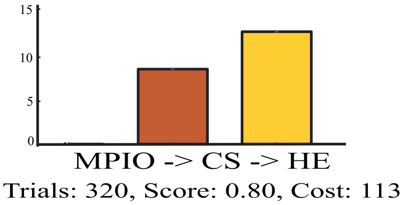


Fig. 8: The performance and cost of each policy trained with a supervisor is reported. The bar graphs shows the breakdown in terms of situations the policy encountered on the test set. The top row corresponds to policies only trained with one supervisor: MPIO, Crowdsourced or Human Expert. The bottom corresponds to policies trained with a hierarchy of supervisors: MPIO and Crowdsource, MPIO and Human Expert, MPIO, Crowdsourced and Human Expert. The final bottom right plot demonstrates the full hierarchy, which achieves the best performance (0.8).

E. Scaling the Hierarchy

The final experiment we ran is to run our algorithm on a hierarchy with 3 supervisors: MPIO, crowdsourced and human expert. We ran each policy until we observed the pay out in terms of reward achieved was not increasing. This results in MPIO for 100 demonstrations, crowdsourced for 120 demonstrations and human expert for 100 demonstrations. Thus resulting in 320 trials on the robot and 320,000 annotated images of what control the robot should apply.

The results of our trained policy, as shown in Fig. 6, demonstrate that by leveraging a hierarchy we are able to achieve a score of 80% with a cost of only 113. We note that policy trained with the human expert supervisor, which incurred a cost of 160, was only able to achieve a score of 0.66%.

While our policy was able to do substantially better, it did score a near successful 40%. These failure modes where do to the robot not being able to accurately manipulate smaller objects, where slight perturbations could result them being pushed into the gripper, and accurately controlling the goal object to land in the circle (i.e. sometimes pushing against the side of the circle instead). We attribute this problem to a limit in the detail of precision our supervisors can currently give, future work will look at more precise analytical methods and fine tuning the policies via self-learning.

VII. Discussions and Future Work

For highly stochastic domains like grasping in clutter, using a hierarchy of supervisors to bootstrap performance at a lower cost seems to be a very sensible thing to do. Our results suggest that we can scale up the amount of data needed to learn the task at a much lower cost than only querying the most-skilled supervisor. However, much work still needs to be done in determining how to appropriately select the next supervisor to receive examples from. While our current strategy worked well empirically, it remains to be seen if this would apply to other domains.

Future work will also look at how to increase the success rate on the grasping in clutter domains. The current failure modes are associated with a lack of fine motor precision. Our current supervisors do not have the ability to give finer motor control corrections, however using more sophisticated analytical methods might enable better performance. Furthermore, applying self-learning to our current policy could be a way to "fine-tune" the current behavior and learn the precision.

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 Steve McKinely and Meng Guo for help in building the hardware setup.

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] 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.

[5] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 41–48.

[6] A. Bicchi and V. Kumar, "Robotic grasping and contact: A review." Citeseer.

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

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

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

[10] 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.

[11] 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.

[12] D. Katz, J. Kenney, and O. Brock, "How can robots succeed in unstructured environments," in *In Workshop on Robot Manipulation: Intelligence in Human Environments at Robotics: Science and Systems*. Citeseer, 2008.

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

[14] J. E. King, J. A. Haustein, S. S. Srinivasa, and T. Asfour, "Nonprehensile whole arm rearrangement planning on physics manifolds," *ICRA 2015 IEEE*.

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

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

[17] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 661–670.

[18] M. Nieuwenhuisen, D. Droeschel, D. Holz, J. Stuckler, A. Berner, J. Li, R. Klein, and S. Behnke, "Mobile bin picking with an anthropomorphic service robot," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 2327–2334.

[19] L. Pinto and A. Gupta, “Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours,” *arXiv preprint arXiv:1509.06825*, 2015.

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

[21] 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.

[22] 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.

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

[24] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 1998.

[25] 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.