Work in Progress: Lifelong Learning for Disturbance Rejection on Mobile Robots

David Isele, José Marcio Luna, Eric Eaton
University of Pennsylvania
{isele, joseluna, eeaton}@seas.upenn.edu

Gabriel V. de la Cruz, James Irwin, Brandon Kallaher, Matthew E. Taylor
Washington State University
{gabriel.delacruz, james.irwin, brandon.kallaher, matthew.e.taylor}@wsu.edu

ABSTRACT

No two robots are exactly the same — even for a given model of robot, different units will require slightly different controllers. Furthermore, because robots change and degrade over time, a controller will need to change over time to remain optimal. This paper leverages lifelong learning in order to learn controllers for different robots. In particular, we show that by learning a set of control policies over robots with different (unknown) motion models, we can quickly adapt to changes in the robot, or learn a controller for a new robot with a unique set of disturbances. Further, the approach is completely model-free, allowing us to apply this method to robots that have not, or cannot, be fully modeled. These preliminary results are an initial step towards learning robust fault-tolerant control for arbitrary robots.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning;
I.2.9 [**Artificial Intelligence**]: Robotics

General Terms

algorithms, experimentation

Keywords

Lifelong Learning, Policy Gradients, Reinforcement Learning, Robotics, Fault-Tolerant Control

1. INTRODUCTION

As robots become more common, there are an increasing number of tasks they will be asked to perform. These tasks may not be specified, or even envisioned, at design time. It is therefore critical that robots be able to learn these task autonomously. *Reinforcement learning* [11, 20] (RL) is one popular method for such autonomous learning, but it may be slow in practice, requiring numerous interactions with the environment to achieve decent performance. Recent work in transfer learning [22] can alleviate some of this burden by using knowledge learned from previous tasks to accelerate learning on a new target task. In this work, we take a *lifelong learning* approach [23], in which the learner faces multiple consecutive tasks and must learn each rapidly by building upon its learned knowledge through transfer, while simultaneously maximizing performance across all known tasks. In particular, we consider a set of similar robots that all have slightly different motion models, each with their own disturbances. This setting is motivated by the inherent differences between robots from small variations in their physical or electrical components.

If the model of the robot was fully known, or could be quickly learned, the dynamics of the system could be stabilized with control theory approaches. However, in many cases such a model is not known, or is complicated enough (or changes quickly enough) that indirect learning of the model is infeasible. Instead, this paper directly learns policies for the different robots through lifelong RL.

Our recent work on lifelong RL [5, 6] has showed that this approach is able to accelerate learning of control policies using policy gradient [21, 26] (PG) methods. Lifelong RL succeeds even when the different systems are encountered consecutively, and it preserves and possibly improves the policies for the earliest encountered tasks (in contrast to transfer methods which typically only optimize performance on the new target system). However, so far this work has been applied only to benchmark problems with known dynamics to demonstrate knowledge sharing, and not yet to more complex robotic control problems. This paper significantly scales up the complexity of experiments by applying lifelong learning techniques to a set of Turtlebot 2 robots, each with their own control disturbances, in the high-fidelity Gazebo simulator. As such, this paper represents an important step to validating lifelong learning on physical robot platforms. The long-term goal of this work is to apply these methods not only to quickly learn controllers for robots with slightly different dynamics, but also to achieve fault-tolerant control by handling minor system failures online.

2. RELATED WORK

Reinforcement learning (RL) is often used to learn controllers in a model-free setting. Amongst RL algorithms, policy gradient methods are popular in robotic applications [12, 17] since they accommodate continuous state/action spaces and can scale well to high dimensional problems. The goal of lifelong learning is to learn a set of policies from consecutive tasks. By exploiting similarities between the tasks,

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), John Thangarajah, Karl Tuyls, Stacy Marsella, Catholijn Jonker (eds.), May 9–13, 2016, Singapore.*

it should be possible to learn the set of tasks much faster than if each task was learned independently. Our previous work showed that lifelong learning could successfully leverage policy gradient methods [5, 6], but had been applied only to simple benchmark dynamical systems and not more complex robotic control problems. There have been some successful examples of lifelong learning on robots, but they tend to focus in skill refinement on a single robot [10, 24] rather than sharing information across multiple robots.

When mathematical models that describe the behavior of physical systems can be constructed, they can be used to analyze, predict and control a robot's behavior. Well-known techniques for modeling physical systems include partial, ordinary differential and difference equations [9, 16], and Discrete Event Systems (DES) such as queuing networks [15, 25] and Petri networks [7]. Typical problems in controlling such systems are regulation, trajectory tracking, disturbance rejection, and robustness [9, 14, 16]. All of these problems are associated with the analysis of the stabilizability of the system, as well as the design of controllers to stabilize it.

Most similar to our setting is that of *disturbance rejection*, where a controller is designed to complete a task while compensating for a disturbance that modifies its nominal dynamics. As long as there is an accurate model of the robot, current methods can handle constant, time-varying, and even stochastic disturbances [8, 9, 14]. However, such methods are generally inapplicable when the robot model is unknown, even if the disturbances are relatively simple. Our work is motivated by control theory approaches, but focuses on leveraging model-free RL techniques.

3. BACKGROUND

This section provides an overview of background material to understand the techniques used in our experiments.

3.1 Reinforcement Learning

An RL agent must sequentially select actions to maximize its expected return. Model-free RL approaches do not require previous knowledge of the system dynamics and control policies are learned directly through the interactions with the system. RL problems are typically formalized as Markov Decision Processes (MDPs) with the form $\langle \mathcal{X}, \mathcal{A}, P, R, \gamma \rangle$ where $\mathcal{X} \subset \mathbb{R}^{d_x}$ is the set of states, $\mathcal{A}$ is the set of actions, $P : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to [0, 1]$ is the state transition probability describing the systems dynamics with initial state distribution P_0, $R : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the reward discount factor. At each time step h, the agent is in the state $\mathbf{x}_h \in \mathcal{X}$ and must choose an action $\mathbf{a}_h \in \mathcal{A}$ so that it transitions to a new state $\mathbf{x}_{h+1}$ with state transition probability $P(\mathbf{x}_{h+1} \mid \mathbf{x}_h, \mathbf{a}_h)$, yielding a reward r_h according to R. The action is selected according to a policy $\pi_{\boldsymbol{\theta}} : \mathcal{X} \times \mathcal{A} \to [0, 1]$, which specifies a probability distribution over actions given the current state and is parameterized by $\boldsymbol{\theta}$. The goal of an RL algorithm is to find an optimal policy π^* that maximizes the expected reward.

PG methods are well suited for solving high dimensional problems with continuous state and action spaces, such as robotic control [17]. The goal of PG is to use gradient steps based on a set of observed state-action-reward trajectories of length H to optimize the expected average return of $\pi_{\boldsymbol{\theta}}$: $\mathcal{J}(\boldsymbol{\theta}) = \int_{\mathbb{T}} p_{\boldsymbol{\theta}}(\tau)\mathcal{R}(\tau)\mathrm{d}\tau$, where $\mathbb{T}$ is the set of all trajectories, $p_{\boldsymbol{\theta}}(\tau) = P_0(\mathbf{x}_0)\prod_{h=0}^{H} p(\mathbf{x}_{h+1} \mid \mathbf{x}_h, \mathbf{a}_h)\pi_{\boldsymbol{\theta}}(\mathbf{a}_h \mid \mathbf{x}_h)$ is the probability of trajectory τ, and $\mathcal{R}(\tau) = \frac{1}{H}\sum_{h=0}^{H} r_h$ is the average per-step reward. Most PG methods (e.g., episodic REINFORCE [26], Natural Actor Critic [17], and PoWER [12]) optimize the policy by maximizing a lower bound on the return, comparing trajectories generated by different candidate policies $\pi_{\hat{\boldsymbol{\theta}}}$. In this particular application, the PG method we use in our experiments is finite differences [11] (FD) which optimizes the return directly.

3.2 Finite Differences for Policy Search

The *Finite Differences* method [11], which has shown past success in robotic control, optimizes the policy $\pi_{\boldsymbol{\theta}}$ directly by computing small changes $\boldsymbol{\Delta\theta}$ in the policy parameters that will increase the expected reward. This process estimates the expected return for each policy parameter variation $(\boldsymbol{\theta}_m + \boldsymbol{\Delta\theta}_p)$ given the sampled trajectories via

$$\Delta\hat{\mathcal{J}}_{\mathbf{p}} \approx \mathcal{J}(\boldsymbol{\theta}_m + \boldsymbol{\Delta\theta}_p) - \mathcal{J}_{ref} \ , \tag{1}$$

where the estimate is taken over n small perturbations in the policy parameters $\{\boldsymbol{\Delta\theta}_p\}_{p=1}^{n}$, the policy parameters at timestep m are given by $\boldsymbol{\theta}_m$, and $\mathcal{J}_{ref}$ is a reference return, which is usually taken as the return of unperturbed parameters $\mathcal{J}(\boldsymbol{\theta})$. The FD gradient method then updates the policy parameters, following the gradient of the expected return $\mathcal{J}$ with a step-size δ, as given by

$$\boldsymbol{\theta}_{m+1} = \boldsymbol{\theta}_m + \delta\nabla_{\boldsymbol{\theta}}\mathcal{J} \ . \tag{2}$$

For efficiency, we can estimate the gradient $\nabla_{\boldsymbol{\theta}}\mathcal{J}$ using linear regression as

$$\nabla_{\boldsymbol{\theta}}\mathcal{J} \approx \left(\boldsymbol{\Delta\Theta}^{\mathsf{T}}\boldsymbol{\Delta\Theta}\right)^{-1}\boldsymbol{\Delta\Theta}^{\mathsf{T}}\Delta\hat{\boldsymbol{J}}_{\boldsymbol{p}} \ , \tag{3}$$

where $\Delta\hat{\boldsymbol{J}}_{\boldsymbol{p}}$ contains all the stacked samples of $\Delta\hat{\mathcal{J}}_{\mathbf{p}}$ and $\boldsymbol{\Delta\Theta}$ contains the stacked perturbations $\boldsymbol{\Delta\theta}_p$. This approach is sensitive to the type and magnitude of the perturbations, as well as to the step size δ. Since the number of perturbations needs to be as large as the number of parameters, this method is considered to be noisy and inefficient for problems with large sets of parameters [11], although we found it to work well and reliably in our setting.

The process is capable of optimizing a policy for a single RL task via repeatedly sampled trajectories (n trajectories for each $m \in \{1, \ldots, M\}$ iteration). In order to share information between different policies that are learned consecutively, we incorporate the PG learning process using FD into a lifelong learning setting, as described next.

4. LIFELONG MACHINE LEARNING

In this section, we describe the framework we use to share knowledge between multiple, consecutive tasks.

4.1 Problem Setting

In the lifelong learning setting [19, 24], the learner optimizes policies for multiple tasks consecutively, rapidly learning each new task policy by building upon its previously learned knowledge. At each time step, the learner observes a task $\mathcal{Z}^{(t)}$, represented as an MDP $\langle \mathcal{X}^{(t)}, \mathcal{A}^{(t)}, P^{(t)}, R^{(t)}, \gamma^{(t)} \rangle$, building on top of the knowledge learned from previous tasks. The task $\mathcal{Z}^{(t)}$ may be new, or it may be a repetition of a known task. After observing T tasks ($1 \leq T \leq T_{max}$), the goal of the learner is to optimize policies for all known tasks $\{\mathcal{Z}^{(1)}, \ldots, \mathcal{Z}^{(T)}\}$ without knowing a priori the total number of tasks T_{max}, their order, or their distribution.

In our application, we use a centralized lifelong learner that is shared between multiple robots; each task corresponds to an RL problem for an individual robot. The policy $\pi_{\boldsymbol{\theta}^{(t)}}$ for task $\mathcal{Z}^{(t)}$ is parameterized by $\boldsymbol{\theta}^{(t)} \in \mathbb{R}^d$. To facilitate transfer between the task policies, we assume there is a shared basis $\boldsymbol{L} \in \mathbb{R}^{d \times k}$ that underlies all policy parameter vectors, and that each $\boldsymbol{\theta}^{(t)}$ can be represented as a sparse linear combination of the basis vectors, given by $\boldsymbol{\theta}^{(t)} = \boldsymbol{L}\boldsymbol{s}^{(t)}$, with coefficients $\boldsymbol{s}^{(t)} \in \mathbb{R}$. Research has shown this factorized model to be effective for transfer in both multi-task [13, 18] and lifelong learning [19] settings.

4.2 Lifelong Learning with Policy Gradients

In our previous work [5], we developed an efficient algorithm for learning in this lifelong setting with policy gradients, known as PG-ELLA. Here, we briefly review this algorithm, which we apply to the multi-robot setting in our experiments. For details, please see the original paper. The one major difference from our previous work is that we employ Finite-Difference methods as the base learner in this paper; our previous work used episodic REINFORCE [26] and natural actor critic [17]. We found FD to be easier to tune and produced better results for our application.

The lifelong learner's goal of optimizing all known policies after observing T tasks is given by the multi-task objective:

$$\underset{\boldsymbol{L},\boldsymbol{S}}{\operatorname{argmin}} \frac{1}{T} \sum_t \left[-\mathcal{J}\left(\boldsymbol{\theta}^{(t)}\right) + \lambda \left\| \boldsymbol{s}^{(t)} \right\|_1 \right] + \mu \|\boldsymbol{L}\|_{\mathsf{F}}^2 \ , \qquad (4)$$

where $\boldsymbol{S} = \left[\boldsymbol{s}^{(1)} \cdots \boldsymbol{s}^{(T)} \right]$ is the matrix of all coefficients, the L_1 norm $\|\cdot\|_1$ enforces sparsity of the coefficients, and the Frobenious norm $\|\cdot\|_{\mathsf{F}}$ regularizes the complexity of $\mathbf{L}$ with regularization parameters $\mu, \lambda \in \mathbb{R}$. To solve Equation 4 efficiently, PG-ELLA: 1.) replaces $\mathcal{J}(\cdot)$ with an upper bound (as done in typical PG optimization), 2.) approximates the first term with a second-order Taylor expansion around an estimate $\boldsymbol{\alpha}^{(t)}$ of the single-task policy parameters for task $\mathcal{Z}^{(t)}$, and 3.) optimizes $\boldsymbol{s}^{(t)}$ only when training on task $\mathcal{Z}^{(t)}$. These steps reduce the learning problem to a series of online update equations that constitute PG-ELLA [5]:

$$\boldsymbol{s}^{(t)} \leftarrow \arg\min_{\boldsymbol{s}} \left\| \boldsymbol{\alpha}^{(t)} - \boldsymbol{L}\boldsymbol{s} \right\|_{\boldsymbol{\Gamma}^{(t)}}^2 + \mu \|\boldsymbol{s}\|_1 \ , \qquad (5)$$

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \left(\boldsymbol{s}^{(t)} \boldsymbol{s}^{(t)\mathsf{T}} \right) \otimes \boldsymbol{\Gamma}^{(t)} \ , \qquad (6)$$

$$\boldsymbol{b} \leftarrow \boldsymbol{b} + \operatorname{vec}\left(\boldsymbol{s}^{(t)} \otimes \left(\boldsymbol{\alpha}^{(t)\mathsf{T}} \boldsymbol{\Gamma}^{(t)} \right) \right) \ , \text{ and} \qquad (7)$$

$$\boldsymbol{L} \leftarrow \operatorname{mat}\left(\left(\frac{1}{T}\boldsymbol{A} + \lambda \boldsymbol{I}_{l \times d_\theta, l \times d_\theta} \right)^{-1} \frac{1}{T}\boldsymbol{b} \right) \ . \qquad (8)$$

where $\|\boldsymbol{v}\|_{\boldsymbol{A}}^2 = \boldsymbol{v}^\top \boldsymbol{A} \boldsymbol{v}$, $\boldsymbol{\Gamma}^{(t)}$ is the Hessian of the PG lower bound on $\mathcal{J}(\boldsymbol{\alpha}^{(t)})$, $\otimes$ is the Kronecker product operator, $\boldsymbol{I}_{m,n}$ is the $m \times n$ identity matrix, and $\boldsymbol{A}$ and $\boldsymbol{b}$ are initialized to be zero matrices. PG-ELLA is given as Algorithm 1.

Algorithm 1 PG-ELLA (k, λ, μ) [5]

$T \leftarrow 0$
$\mathbf{A} \leftarrow \mathbf{zeros}_{k \times d, k \times d}$, $\mathbf{b} \leftarrow \mathbf{zeros}_{k \times d, 1}$
$\boldsymbol{L} \leftarrow \text{RandomMatrix}_{d,k}$
while some task $\mathcal{Z}^{(t)}$ is available **do**
  if isNewTask($\mathcal{Z}^{(t)}$) **then**
    $T \leftarrow T + 1$
    $\left(\mathbb{T}^{(t)}, R^{(t)}\right) \leftarrow \text{getRandomTrajectories}()$
  else
    $\left(\mathbb{T}^{(t)}, R^{(t)}\right) \leftarrow \text{getTrajectories}\left(\boldsymbol{\alpha}^{(t)}\right)$
    $\mathbf{A} \leftarrow \mathbf{A} - \left(\boldsymbol{s}^{(t)}\boldsymbol{s}^{(t)\mathsf{T}}\right) \otimes \boldsymbol{\Gamma}^{(t)}$
    $\mathbf{b} \leftarrow \mathbf{b} - \text{vec}\left(\boldsymbol{s}^{(t)\mathsf{T}} \otimes \left(\boldsymbol{\alpha}^{(t)\mathsf{T}}\boldsymbol{\Gamma}^{(t)}\right)\right)$
  end if
  Compute $\boldsymbol{\alpha}^{(t)}$ and $\boldsymbol{\Gamma}^{(t)}$ from $\mathbb{T}^{(t)}$ using PG
  $\boldsymbol{s}^{(t)} \leftarrow \arg\min_{\boldsymbol{s}} \left\|\boldsymbol{\alpha}^{(t)} - \boldsymbol{L}\boldsymbol{s}\right\|_{\boldsymbol{\Gamma}^{(t)}}^2 + \mu \|\boldsymbol{s}\|_1$
  $\mathbf{A} \leftarrow \mathbf{A} + \left(\boldsymbol{s}^{(t)}\boldsymbol{s}^{(t)\mathsf{T}}\right) \otimes \boldsymbol{\Gamma}^{(t)}$
  $\mathbf{b} \leftarrow \mathbf{b} + \text{vec}\left(\boldsymbol{s}^{(t)\mathsf{T}} \otimes \left(\boldsymbol{\alpha}^{(t)\mathsf{T}}\boldsymbol{\Gamma}^{(t)}\right)\right)$
  $\mathbf{L} \leftarrow \text{mat}\left(\left(\frac{1}{T}\mathbf{A} + \lambda \mathbf{I}_{k \times d, k \times d}\right)^{-1} \frac{1}{T}\mathbf{b}\right)$
  for $t \in \{1, \ldots, T\}$ **do**: $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{L}\boldsymbol{s}^{(t)}$
end while

5. DISTURBANCE REJECTION FOR ROBOTICS VIA LIFELONG LEARNING

This paper's goal is to present our progress adapting PG-ELLA to learn policies for robotic control, using simulated Turtlebot 2's in ROS. In our previous work, PG-ELLA was only ever evaluated on the control of simple dynamical systems with well-known models, such as inverted pendulums.

Specifically, we focus on the well-known problem of *disturbance rejection* in robotics. In disturbance rejection, it is assumed that the nominal dynamics of the plant (i.e., system) are additively disturbed by a signal $\boldsymbol{\omega}$. The system dynamics are given by $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{\omega}$, where $\boldsymbol{f} : \mathbb{R}^{d_x} \times \mathbb{R} \rightarrow \mathbb{R}^{d_x}$, and $\boldsymbol{\omega} \in \mathbb{R}^{d_x}$. The goal is to determine the control input that minimizes the effect of the disturbance in the return function, so that the plant can execute this task.

There are well-known optimal control [8, 14] techniques to solve this problem, if there is an available mathematical model. However, if such a model is not available, or there is a partial knowledge of the model, formal solutions are not effective. RL offers one alternative solution to this problem, but in a single-task setting, it would require numerous interactions with the environment to learn an effective control policy to compensate for the disturbance. However, in a lifelong learning setting, the learner could build upon its existing knowledge in controlling other systems (each with their own disturbances) to rapidly learn a control for a system with a novel disturbance. We assume that the learner attempts to optimize control policies for a set of robots, all of which have the same nominal dynamics, given by $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$. Each robot is affected by a different disturbance function $\boldsymbol{\omega}^{(t)}$. All $\boldsymbol{\omega}^{(t)}$'s share the same structure but different parameters, *e.g.,* all the disturbances are constant but different, are sinusoidal with different phases or amplitudes, etc.

Lifelong machine learning takes advantage of the potential for knowledge transfer among different tasks. After learning how to compensate for the disturbance without requiring a mathematical model, a general structure of the policy can be proposed. Then a reward function is designed so that the lifelong learner penalizes the effect of the disturbance over either a realistic simulated robot or an actual one. In the next section, we present our preliminary application of lifelong learning to this problem of robotic control under disturbances.

6. EXPERIMENTS

This section describes our initial experiments applying lifelong learning to the problem of disturbance rejection for robotics, using the Turtlebot 2 platform [3] (Figure 1a). In order to simulate a wide variety of Turtlebots, each with their own disturbances, we conducted the experiments using the high-fidelity Gazebo simulator [1, 2]. However, our experimental setup allows us to use the same code in both simulation and on physical Turtlebots. The implementation of our approach uses Python and the Hydro version of ROS.

In our disturbance rejection scenario, we focused on learning control policies for driving the Turtlebots to a goal location as they experience disturbances in their wheel actuators. This disturbance emulates a bias on the angular velocity of each robot that forces the robot to compensate for the induced failure. Note that this type of disturbance in actuation is common in physical robots and autonomous ground vehicles, stemming from a variety of sources, such as calibration issues, wear in the drive train, or interference from debris. To simulate these disturbances, we induce a random and constant disturbance to the control signal that is drawn uniformly from $[-0.1, 0.1]$ and measured in m/s for each robot. These limits were selected to provide a large noise that was still within the bounds of the Turtlebot control system. Although we use a constant difference for now, the difficulty of the learning problem can easily be increased later by introducing time-varying stochastic disturbances.

We assume little knowledge of the Turtlebot's dynamics. In our application, each robot's state is defined as $\boldsymbol{x} = (\rho, \gamma, \psi)^\mathsf{T}$, with ρ, γ and ψ as illustrated in Fig. 1b. To extract state features for learning, we use the following nonlinear transformation of the position and heading angle:

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{pmatrix} \rho\cos(\gamma) \\ \gamma \\ \frac{\cos(\gamma)\sin(\gamma)}{\gamma}(\gamma + \psi) \\ 1 \end{pmatrix} . \tag{9}$$

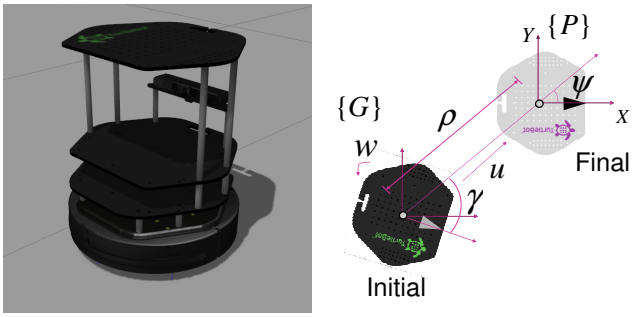
Given the stochastic policy $\pi_{\boldsymbol{\theta}^{(t)}} \sim \mathcal{N}(\boldsymbol{a}^{(t)}, \boldsymbol{\Sigma})$ for the t-th Turtlebot, the control action is then specified by $\boldsymbol{a}^{(t)} = \boldsymbol{\theta}^{(t)^\mathsf{T}}\boldsymbol{\phi}(\boldsymbol{x}) = (u, w)^\mathsf{T}$ where u and w are the linear and angular velocities of the robots. This particular choice of nonlinear transformation is inspired by a simplified kinematic model for unicycle-like vehicles in polar coordinates [4]. In this model, the state space is given by $\mathcal{X} \subset \mathbb{R}^3$ and the action space is described by $\mathcal{A} \subset \mathbb{R}^2$. This simplified kinematics model ignores contributions to the dynamics of the system from the robot's mass, damping and friction coefficients, as well as inputs such as forces and torques.

In these preliminary experiments, we use FD [11] as the base learner in PG-ELLA for its simplicity and good performance in simulation, despite its known stability issues (which we did not experience). In future work, we plan to compare this approach with different base learners, such as natural actor critic [17] and episodic REINFORCE [26].

6.1 Methodology

We generated 20 simulated Turtlebots, each with a different constant disturbance and a unique goal, both selected uniformly. This number of robots provided a large task diversity, while still being small enough to simulate practically.

We used FD as our PG method to train each robot's initial policy for $M = 20$ iterations with $n = 15$ roll-outs per iteration and $H = 50$ time steps per roll-out. If the robot reached the goal in less than 50 time steps, the experiment continues to run to completion ensuring that a good policy reaches the goal and stops. These initial policies were then used as the $\boldsymbol{\alpha}^{(t)}$'s for PG-ELLA. Note that all systems in our experiment require more than 20 iterations to converge to a good controller, so subsequent policy improvement is essential for decent performance. The number of roll-outs and time steps were selected to allow for successful learning while minimizing the runtime.



(a) Turtlebot (b) State variables

Figure 1: (a) The Turtlebot 2 model in Gazebo, and (b) its state variables in the simplified go-to-goal problem.

PG-ELLA trains the shared knowledge repository $\boldsymbol{L}$ and sparse policy representations $\boldsymbol{s}^{(t)}$ using the update equations given by Equations 5–8. Tasks were encountered randomly with repetition and learning stopped once every task was observed once. For our experiments, we approximate the Hessian with the identity matrix because it was found to work well in practice and reduced the number of rollouts. For the parameters unique to PG-ELLA, we use $k = 8$ columns in the shared basis, and use sparsity coefficient $\mu = 1 \times 10^{-3}$ and regularization coefficient $\lambda = 1 \times 10^{-8}$. These coefficients were tuned manually, and were found to be relatively easy to tune, being largely insensitive to changes within an order of magnitude. It is worth noting that similar performance was shown for $k \in \{4, \ldots, 12\}$ and $k = 8$ was selected as the mean. The learning rate was set to $\delta = 1 \times 10^{-6}$ and the standard deviation of the policy was set to $\sigma = 0.001$.

Figure 2 compares the reward for policies learned by PG-ELLA against PG, averaged over all 20 robots over 6 simulation trials. We start measuring performance at 20 iterations, since the initial seed policies for PG-ELLA were learned using those first 20 iterations; we then plot the learning curves as the polices are improved by either FD or PG-ELLA for an additional 80 learning iterations. We see that PG-ELLA is successfully able to reconstruct the control policies and provide a slight improvement in performance through positive transfer. Figure 3 depicts the gain in reward, showing positive transfer between tasks. Although these preliminary results show only a slight improvement currently, we suspect further refinements will enable us to achieve larger transfer.

7. CONCLUSIONS

We demonstrate the use of lifelong learning for disturbance rejection on Turtlebots. This preliminary work is intended to lay the foundation for fault-tolerant control in multi-agent systems. The results show that PG-ELLA can

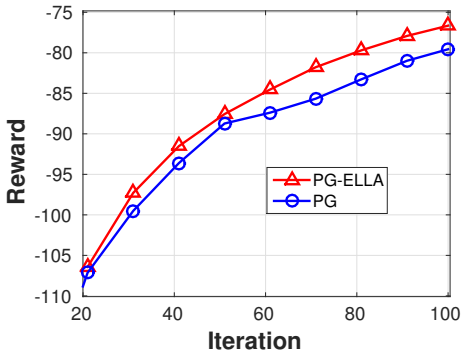


Figure 2: Learning curves for PG and PG-ELLA using a finite-difference base learner. Using PG-ELLA to transfer information between tasks improves performance over PG.

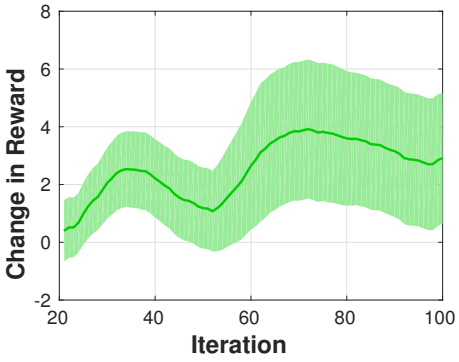


Figure 3: The positive transfer achieved by lifelong learning.

be successfully implemented on simulated and complex 3D environments, yielding an improvement over standard PG methods. This suggests that PG-ELLA can benefit real robotic systems. The implementation on real Turtlebots and quadrotors is part of our future research agenda.

Acknowledgments

Research at Penn was partially supported by grants ONR N00014-11-1-0139 and AFRL FA8750-14-1-0069. Research at Washington State University was supported in part by grants AFRL FA8750-14-1-0069, AFRL FA8750-14-1-0070, NSF IIS-1149917, NSF IIS-1319412, USDA 2014-67021-22174, and a Google Research Award.

REFERENCES

[1] Gazebo. `http://gazebosim.org/`, 2016.

[2] Ros.org: Powering the world's robots. [Online]: `http://www.ros.org/`, 2016.

[3] Turtlebot 2. `http://www.turtlebot.com/`, 2016.

[4] M. Aicardi, G. Casalino, A. Balestrino, & A. Bicchi. Closed loop smooth steering of unicycle-like vehicles. In *Proc. of the IEEE Conference on Decision and Control*, pp. 2455–2458, 1994.

[5] H. Bou Ammar, E. Eaton, & P. Ruvolo. Online multi-task learning for policy gradient methods. *International Conference on Machine Learning*, 2014.

[6] H. Bou Ammar, E. Eaton, P. Ruvolo, & M. E. Taylor. Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment. *International Joint Conference on Artificial Intelligence*, 2015.

[7] C. G. Cassandras & S. Lafortune. *Introduction to Discrete Event Systems*, 2nd ed. Springer, NY, 2008.

[8] P. Dorato, C. Abdallah, & V. Cerone. *Linear Quadratic Control*. Krieger, 2000.

[9] H. Khalil. *Nonlinear Systems*. Prentice Hall, 2002.

[10] A. Kleiner, M. Dietl, & B. Nebel. Towards a life-long learning soccer agent. In *RoboCup 2002: Robot Soccer World Cup VI*, pp. 126–134. Springer, 2002.

[11] J. Kober, J. A. Bagnell, & J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 2013.

[12] J. Kober & J. Peters. Policy search for motor primitives in robotics. *Advances in Neural Information Processing Systems*, pp. 849–856, 2009.

[13] A. Kumar & H. Daume III. Learning task grouping and overlap in multi-task learning. *International Conference on Machine Learning*, 2012.

[14] F. L. Lewis & V. L. Syrmos. *Optimal Control*. John Wiley & Sons, 3rd edition, 2012.

[15] J. M. Luna, C. T. Abdallah, & G. Heileman. Performance optimization and regulation for multitier servers. In *Proc. of IEEE International Conference on Decision and Control*, pp. 1026–1032, 2015.

[16] N. S. Nise. *Control Systems Engineering*. John Wiley & Sons, 7th edition, 2010.

[17] J. Peters & S. Schaal. Natural actor-critic. *Neurocomputing*, 71(7):1180–1190, 2008.

[18] B. Romera-Paredes, H. Aung, N. Bianchi-Berthouze, & M. Pontil. Multilinear multitask learning. *International Conference on Machine Learning*, pp. 1444–1452, 2013.

[19] P. Ruvolo & E. Eaton. ELLA: An efficient lifelong learning algorithm. *International Conference on Machine Learning*, 28:507–515, 2013.

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

[21] R. S. Sutton, D. A. McAllester, S. P. Singh, & Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 99:1057–1063, 1999.

[22] M. E. Taylor & P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

[23] S. Thrun. Is learning the n-th thing any easier than learning the first? *Advances in Neural Information Processing Systems*, pp. 640–646, 1996.

[24] S. Thrun & T. M. Mitchell. *Lifelong robot learning*. Springer, 1995.

[25] B. Urgaonkar, G. Pacifi, P. Shenoy, M. Spreitzer, & A. Tantawi. Analytic modeling of multitier internet applications. *ACM Trans. on the Web*, 1(1), May 2007.

[26] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.