Lifelong Learning for Disturbance Rejection on Mobile Robots

David Isele, José Marcio Luna, Eric Eaton,
Gabriel V. de la Cruz, James Irwin, Brandon Kallaher and Matthew E. Taylor

***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. Furthermore, the approach is completely model-free, allowing us to apply this method to robots that have not, or cannot, be fully modeled.**

I. 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* (RL) [1], [2] 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* [3] 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 [4], in which the learner faces multiple consecutive tasks and must learn each rapidly by building upon its learned knowledge, while simultaneously maximizing performance across all known tasks. In particular, we consider a set of similar robots that all have slightly different motion models, but with varying 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, is complicated enough or is subject to unpredictable changes 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 shown that this approach is able to accelerate learning of the control policies for a variety of dynamical systems using policy gradient [7], [8] (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. This is in contrast to transfer methods which typically only optimize performance on the new target system. However, 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 [9] and AR.Drone [10] robots, each with their own control disturbances, in both the high-fidelity Gazebo simulator and on real robotic platforms. As such, this paper represents a milestone in validating lifelong learning on physical robotic platforms.

II. RELATED WORK

Reinforcement learning (RL) is often used to learn controllers in a model-free setting. Among RL algorithms, policy gradient methods are popular in robotic applications [11], [12] 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 leveraging similarities between the tasks, 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 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 on skill refinement for a single robot [13], [14], [15] 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 [16], [17], and Discrete Event Systems (DES) such as queuing networks [18] and Petri networks [19]. Typical problems in controlling such systems are regulation, trajectory tracking, disturbance rejection, and robustness [16], [17], [20]. All of these problems are associated with the analysis of the stabilizability of

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

D. Isele, J.M. Luna and E. Eaton are associated with the Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104, USA {isele,joseluna,eeaton}@seas.upenn.edu

G.V. de la Cruz, J. Irwin, B. Kallaher and M.E. Taylor are associated with the School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164, USA {gabriel.delacruz, james.irwin, brandon.kallaher}@wsu.edu, taylorm@eecs.wsu.edu

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 [16], [20], [21]. 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.

III. Background

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

A. 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, $R : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ 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 $P(\mathbf{x}_{h+1}|\mathbf{x}_h, \mathbf{a}_h)$, yielding a reward r_h according to R. The action is selected according to a policy $\pi : \mathcal{X} \times \mathcal{A} \to [0, 1]$, which specifies a probability distribution over actions given the current state. 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 [12]. The goal of PG is to use gradient steps to optimize the expected average return, given by $\mathcal{J}(\boldsymbol{\theta}) = \int_{\mathbb{T}} p_{\boldsymbol{\theta}}(\tau)\mathcal{R}(\tau)\mathrm{d}\tau$, where $\mathbb{T}$ is the set of all trajectories, $p_{\boldsymbol{\theta}}(\tau) = \prod_{h=0}^{H} p(\mathbf{x}_{h+1}|\mathbf{x}_h, \mathbf{a}_h)\pi(\mathbf{a}_h, \mathbf{x}_h)$ is the probability of trajectory τ, and $\mathcal{R}(\tau) = \frac{1}{H}\sum_{h=0}^{H} r(\mathbf{s}_h, \mathbf{a}_h, \mathbf{s}_{h+1})$ is the average per-step reward. Most PG methods (*e.g.,* episodic REINFORCE [8], Natural Actor Critic [12], and PoWER [11]) optimize the policy by maximizing a lower bound on the return, comparing trajectories generated by different candidate policies π. In this particular application, the PG method we use in our experiments is finite differences [1] (FD) which optimizes the return directly.

B. Finite Differences for Policy Search

The *Finite Differences* method [1], 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 time step 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 [1], 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 *i.e.,* 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.

IV. Lifelong Machine Learning

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

A. Problem Setting

In the lifelong learning setting [14], [22], the learner optimizes policies for multiple tasks consecutively, rapidly learning each new task policy by building upon its previously learned knowledge. At each round $t \in \{1, \ldots, T_{max}\}$, the learner observes a task $\mathcal{Z}^{(t)}$, represented as an MDP with the form $\langle \mathcal{X}^{(t)}, \mathcal{A}^{(t)}, P^{(t)}, R^{(t)}, \gamma^{(t)} \rangle$, building on top of the knowledge learned from previous tasks $\{\mathcal{Z}^{(1)}, \ldots, \mathcal{Z}^{(t-1)}\}$. The goal of the learner is to optimize policies for all tasks $\{\mathcal{Z}^{(1)}, \ldots, \mathcal{Z}^{(T_{max})}\}$ 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 $\boldsymbol{s}^{(t)} \in \mathbb{R}^l$. Research has shown this factorized model to be effective for transfer in both multi-task [23], [24] and lifelong learning [22] settings.

B. 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 is that we employ Finite-Difference methods as the base learner in this paper; our previous work used episodic REINFORCE [8] and natural actor critic [12].

The lifelong learner's goal of optimizing the policies for all known tasks after observing task $\boldsymbol{s}^{(T)}$ can be formulated as the following multi-task objective:

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

where $\boldsymbol{S}=\left[\boldsymbol{s}^{(1)}\cdots\boldsymbol{s}^{(T)}\right]$ is the matrix of all coefficients, the L1 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 (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 on-line update equations that constitute PG-ELLA [5]:

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

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

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

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

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**
    $T\leftarrow T+1$
    $\left(\mathbb{T}^{(t)},R^{(t)}\right)\leftarrow$ getRandomTrajectories()
  else
    $\left(\mathbb{T}^{(t)},R^{(t)}\right)\leftarrow$ 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
  $\boldsymbol{s}^{(t)}\leftarrow\arg\min_{\boldsymbol{s}}\left\|\boldsymbol{\alpha}^{(t)}-\boldsymbol{L}\boldsymbol{s}^{(t)}\right\|_{\boldsymbol{\Gamma}^{(t)}}^2+\mu\left\|\boldsymbol{s}\right\|_1$
  $\mathbf{A}\leftarrow\mathbf{A}+\left(\boldsymbol{s}^{(t)}\boldsymbol{s}^{(t)\mathsf{T}}\right)\otimes\boldsymbol{\Gamma}^{(t)}$
  $\mathbf{b}\leftarrow\mathbf{b}+\text{vec}\left(\boldsymbol{s}^{(t)\mathsf{T}}\otimes\left(\boldsymbol{\alpha}^{(t)\mathsf{T}}\boldsymbol{\Gamma}^{(t)}\right)\right)$
  $\mathbf{L}\leftarrow\text{mat}\left(\left(\frac{1}{T}\mathbf{A}+\lambda\mathbf{I}_{k\times d,k\times d}\right)^{-1}\frac{1}{T}\mathbf{b}\right)$
  for $t\in\{1,\ldots,T\}$ **do**: $\boldsymbol{\theta}^{(t)}\leftarrow\boldsymbol{L}\boldsymbol{s}^{(t)}$
end while

V. Disturbance Rejection for Robotics via Lifelong Learning

This paper's goal is to adapt PG-ELLA to learn policies for robotic control, using both simulated and real Turtlebots, as well as simulated AR.Drones. 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 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 [20], [21] 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 $\boldsymbol{\omega}^{(t)}$ share the same structure but different parameters, *e.g.,* all the disturbances are constant but different, all are sinusoidal with different phases or amplitudes, etc.

Lifelong machine learning takes advantage of the potential for knowledge transfer among the different tasks, learning how to compensate for the disturbance without requiring a mathematical model describing the system dynamics. 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 application of lifelong learning to this problem of robotic control under disturbances.

VI. Experimental Design

In our disturbance rejection scenario, we focused on learning control policies for navigating vehicles to a goal location as they experience disturbances in their actuators. We cover two case studies using two different commercial robotic platforms: the Turtlebot 2 [9] and the AR.Drone

quadrotor [10]. We emulate disturbances in their actuators, namely, wheel servos and propellers, respectively. Each case study involves multiple robots from a single platform, and the disturbances are assumed to be different for each robot. The main goal in both case studies is to build a knowledge base from previously learned policies that allows each robot to compensate for the disturbance and fulfill its goal, while transferring this knowledge to new robots with different disturbances. This section describes the validation procedure of our lifelong machine learning approach. In order to obtain the simulation and experimental results we use the Hydro version of the Robotic Operating System (ROS) [25].

A. Case Study: Disturbance Rejection on Turtlebots

In this scenario, each Turtlebot faces a disturbance consisting of a bias on its angular and linear velocity, forcing the robots to compensate for the induced failure to navigate successfully. 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 $[-0.1, 0.1] \subset \mathbb{R}$ for each robot and measured in m/s. These limits were selected to provide a large noise that was within the bounds of the Turtlebot control system.

To simulate lifelong learning on numerous Turtlebots, each with their own disturbances, we conducted simulations using the high-fidelity Gazebo simulator [26], [25]. To show that lifelong learning is similarly effective on real robots, we also evaluated our approach on five real Turtlebots.

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 [27]. 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.

1) Simulation Methodology: In these experiments, we use FD [1] 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). 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.

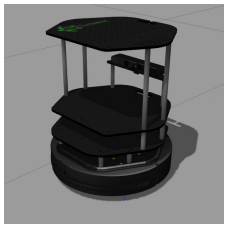
To evaluate the system performance, we use FD as our PG method to train 19 robots initial policies for $M = 100$ iterations with $n = 21$ roll-outs per iteration and $H = 70$ time steps per roll-out. If the robot reached the goal in less than 70 time steps, the experiment continues to run to completion ensuring that a good policy reaches the goal and stops. The 20th robot is initialized with the mean policy of the observed tasks and trained for only $M = 10$ iterations. Note that all systems in our experiment require more than 10 iterations to converge to a good controller, so subsequent policy improvement is essential for decent performance. These policies are then used as the $\boldsymbol{\alpha}^{(t)}$ estimates for PG-ELLA. The number of roll-outs and time steps were selected to allow for successful learning while minimizing the runtime.

PG-ELLA trains the shared knowledge repository $\boldsymbol{L}$ and sparse policy representations $\boldsymbol{s}^{(t)}$ using the update equations given by (5)–(8). 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 roll-outs. For PG-ELLA's parameters, we use $k = 4$ columns in the shared basis, sparsity coefficient $\mu = 1 \times 10^{-5}$, and regularization coefficient $\lambda = 1 \times 10^{-3}$. The learning rate was set to $\delta = 1 \times 10^{-6}$ and the policy's standard deviation was $\sigma = 0.001$.

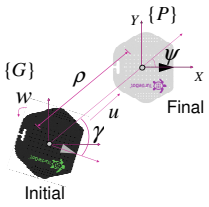
2) Application to Physical Robots: To demonstrate our method on a real Turtlebot, we learned four tasks using conventional PG and then we evaluate the transfer to a fifth task. We reduce the number of roll-outs to $n = 11$ and the number of learning iterations to $H = 30$ to minimize the experimentation time. In contrast to simulation, physical robots have battery limitations, and the continuous generation of random trajectories require a human presence to guarantee correct execution of the experiments. All other parameters, including the added noise, are kept the same as the simulation. To get position information, a beacon which can easily be segmented from camera data is used to mark the goal. The Turtlebot is equipped with a Kinect whose depth information provides the angle and distance. If the robot loses sight of the goal, it enters a recovery mode which rotates it toward the last known goal location, and a penalty is applied for each time step spent in recovery mode. With the fewer number of iterations, only one of the four Turtlebots used as source tasks reached the goal. However, all four systems did exhibit learning improvement. We showcase the performance of the physical robots in a video submitted in parallel to IROS.

3) Turtlebot Results: Fig. 2 compares the reward for policies learned by PG-ELLA on the 20th task against PG, averaged over 20 trials. We begin measuring performance for PG-ELLA at 10 iterations, since the initial seed policies for PG-ELLA were learned using those first 10 iterations. We then plot the learning curves as the polices are improved by PG for an additional 90 learning iterations.

These results show that PG-ELLA is able to both successfully reconstruct and improve the control policies through positive transfer with respect to PG. This is clearly shown



(a) Turtlebot



(b) State variables

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

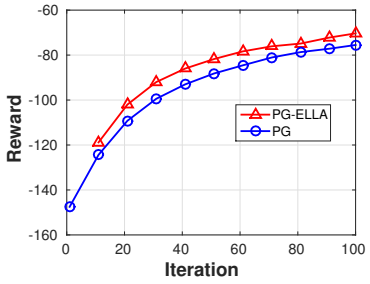


Fig. 2: Learning curves for PG and PG-ELLA in the Turtlebot simulation, averaged over 20 trials.

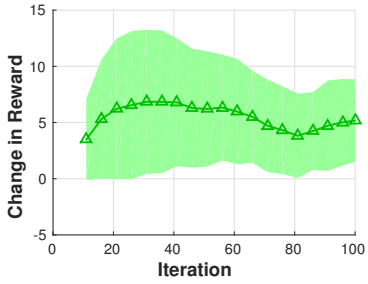


Fig. 3: Change in reward as a measure of positive transfer achieved by lifelong learning on Turtlebots.

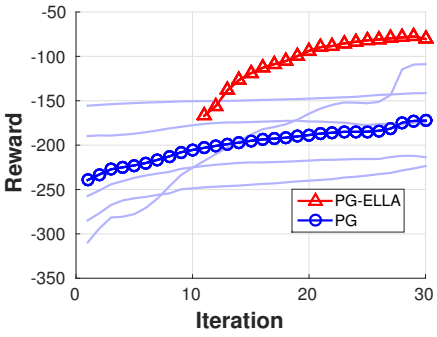


Fig. 4: Learning curves on the real Turtlebots. PG-ELLA (red line) on a new Turtlebot performs better than PG on five other systems (light blue line) and their mean (dark blue line).

by both the initial "jump-start" improvement at the 10th iteration, and the consistently better learning performance over the first 100 iterations. Fig. 3 depicts the change in reward over the learning curve, showing positive transfer between tasks. After only 10 iterations to initialize $\boldsymbol{\alpha}^{(t)}$, we obtain improved initial performance and better reward on PG-ELLA via transfer from the shared knowledge base.

Fig. 4 shows the results of learning on real Turtlebots with the effect of disturbances, confirming the improvement from PG-ELLA on physical systems. These results compare the performance of PG-ELLA (red line) on a new Turtlebot to the performance of PG on four systems (light blue lines) and their mean (dark blue line). In particular, the PG policy exhibiting the best performance is the robot that was not affected by any disturbance, yet is still out-performed by PG-ELLA. These results also show that PG is negatively affected by the disturbances in many cases, while PG-ELLA exhibits better and more consistent learning.

One possible explanation for this is that PG's learning is hindered on the real Turtlebots due to the noisy environment of the physical robots, the intricacies of the empirical reward function, and the presence of local optima in the objective function. By sharing knowledge from other tasks, PG-ELLA is able to compensate for these aspects, and exhibit more rapid learning—this hypothesis agrees with our results on the larger set of simulated Turtlebots.

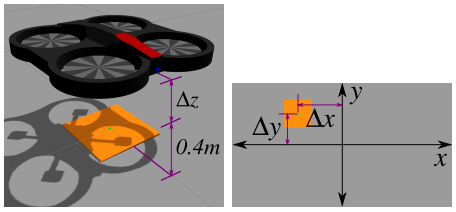
B. Case Study: Disturbance Rejection on AR.Drones

Aerial vehicles are a compelling domain for testing our approach. We use the AR.Drone quadrotor and, as in the Turtlebot case study, we evaluated our approach using the TUM AR.Drone Simulator [28] and Gazebo in order to simulate lifelong learning over numerous quadrotors, each with their own disturbance. For each quadrotor, our approach learns a policy to land the AR.Drone on a marker, while the quadrotor is affected by a disturbance in its propellers that bias its motion. The disturbance is emulated by a bias on the linear velocities in the x and y coordinates, namely v_x and v_y, as may commonly be caused by wind or wear on the rotors. To simulate these disturbances, we induced a random and constant disturbance drawn uniformly from $[-0.15, 0.15] \subset \mathbb{R}$ for each robot and measured in m/s.

We localize the quadrotor using its downward-facing camera with the assumption that the landing pad marker is visible. Each robot's state is defined as $\boldsymbol{x} = (x, y, z)^\mathsf{T}$, where x and y are the Cartesian coordinates of the goal relative to the center of the camera as the origin. The difference between the quadrotor's altitude and the landing altitude is given as z, measured using the robot's on-board sonar height sensor. The state features for learning are the normalized values of the robot's state. The normal range of values for x and y are ± 2 and ± 1.5, respectively. If the landing pad marker is not visible, the (x, y) state is represented as $x_h = 4$ and $y_h = 3$, which is simply double the normal maximum states. During the learning phase, at time-step h, when the quadrotor is at $z_h = 0$ ($0.4\ m$ in altitude) and the landing pad marker is at $x_h = [-0.7, 0.7]$ and $y_h = [-0.8, 0.6]$, then the landing action is triggered. There is an offset in y since the downward-facing camera underneath the quadrotor is not centered. The landing altitude of $0.4\ m$ was set because, based on the specifications, this is the lowest altitude possible that the sonar sensor accurately measures, and therefore considered a good altitude to land from.

Landing occurs regardless of the quadrotor's orientation, so the system send signals in the format of a ROS Twist message. It is composed of directives to control the linear velocity of the quadrotor in the x, y, and z directions. Valid actions are in the range of $\pm 1 m/s$ for each axis of control.

Two safety features were added to ensure that the quadro-



(a) Z state (b) X,Y state

Fig. 5: (a) The AR.Drone 2 model in Gazebo, and (b) downward-facing camera feed used for localization.

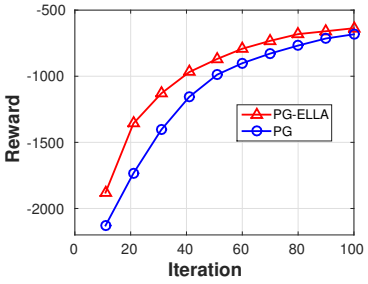


Fig. 6: Learning curves for PG and PG-ELLA in the AR.Drone simulation, averaged over 20 trials.

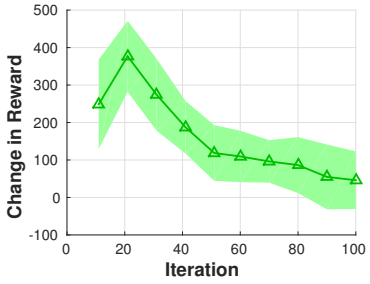


Fig. 7: Change in reward as a measure of positive transfer achieved by lifelong learning on AR.Drones.

tor does not get damaged when the learned policy is applied to a physical robot: 1) when the landing marker is out of sight, the quadrotor is set to hover mode, and 2) with a maximum height sensor range of $3.0m$, when $z_h \geq 2.4m$, the system overrides the action by $-0.1m/s$, forcing a downward elevation. These safety features are not needed for our simulation; this was done in anticipation of future work involving a physical AR.Drone, to make the simulated AR.Drone more similar to the real quadrotor.

1) Simulation Methodology: We generated 20 simulated quadrotors, each with a different constant disturbance selected uniformly. Using FD as our PG learner, we train 19 robot's initial policies for $M = 100$ iterations with $n = 15$ roll-outs per iteration and $H = 150$ time steps per roll-out. The 20th robot is initialized with the mean policy over the 19 observed policies and is trained for only $M = 10$ iterations. However, out of the 19 robots with 100 learning iterations, 3 systems did not converge to a good controller.

We follow the same methodology as for the Turtlebot to train PG-ELLA, setting $k = 4$, $\mu = 1 \times 10^{-5}$, and $\lambda = 1 \times 10^{-6}$. The learning rate was $\delta = 1 \times 10^{-6}$ and the policy's standard deviation was set to $\sigma = 0$ to make it deterministic.

2) AR.Drone Results: Since the 20th task for PG-ELLA was reconstructed from a policy after 10 iterations, we start comparing PG-ELLA with PG after those 10 iterations, then continue learning for more 90 iterations using PG. Fig. 6 shows the learning curve for policies trained by PG-ELLA (red line) against those learned by PG (blue line), averaged over 20 tasks. The results show that PG-ELLA is also effective in the quadrotor domain for both reconstructing and improving the learned policies. Similar to the Turtlebots, we see improved initial "jump-start" performance and more rapid learning than PG. Fig. 7 depicts this gain in reward.

VII. Conclusions

We showed that lifelong learning is effective for disturbance rejection on Turtlebots and AR.Drones, in both simulated 3D environments and on real robots, outperforming single-task learning. This work is intended to lay the foundation for fault-tolerant control on multi-agent systems based on knowledge learned from previous experience.

References

[1] J. Kober, J. A. D. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *Int. J. of Robotics Research*, July 2013.

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

[3] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *J. Mach. Learning Research*, vol. 10, pp. 1633–1685, 2009.

[4] S. Thrun, "Is learning the n-th thing any easier than learning the first?" *Advances in Neural Inform. Process. Syst.*, pp. 640–646, 1996.

[5] H. Bou Ammar, E. Eaton, and P. Ruvolo, "Online multi-task learning for policy gradient methods," *Int. Conf. on Mach. Learning*, 2014.

[6] H. Bou Ammar, E. Eaton, J. M. Luna and P. Ruvolo, "Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning," *Int. Joint Conf. on Artificial Intell.*, 2015.

[7] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Neural Inform. Process. Syst.*, vol. 99, pp. 1057–1063, 1999.

[8] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[9] "Turtlebot 2," [Online]. Available: www.turtlebot.com, 2016.

[10] "Parrot AR.Drone 2," [Online]. Available: ardrone2.parrot.com, 2016.

[11] J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Advances in Neural Inform. Process. Syst.*, pp. 849–856, 2009.

[12] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7, pp. 1180–1190, 2008.

[13] A. Kleiner, M. Dietl and B. Nebel, "Towards a life-long learning soccer agent," in *RoboCup 2002: Robot Soccer World Cup VI*. Springer, 2002.

[14] S. Thrun and T. M. Mitchell, *Lifelong robot learning*. Springer, 1995.

[15] A. White, J. Modayil, and R. S. Sutton, "Scaling life-long off-policy learning," in *IEEE Int. Conf. on Development and Learning and Epigenetic Robotics (ICDL)*. IEEE, 2012, pp. 1–6.

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

[17] N. S. Nise, *Control Systems Engineering*, 7th ed. Wiley & Sons, 2010.

[18] B. Urgaonkar, G. Pacifi, P. Shenoy, M. Spreitzer, and A. Tantawi, "Analytic modeling of multitier internet applications," *ACM Trans. on the Web*, vol. 1, no. 1, May 2007.

[19] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. New York: Springer, 2008.

[20] F. Lewis & V. Syrmos, *Optimal Control*, 3rd ed. Wiley & Sons, 2012.

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

[22] P. Ruvolo and E. Eaton, "ELLA: An efficient lifelong learning algorithm," *Int. Conf. on Mach. Learning*, vol. 28, pp. 507–515, 2013.

[23] A. Kumar and H. Daume III, "Learning task grouping and overlap in multi-task learning," *Int. Conf. on Mach. Learning*, 2012.

[24] B. Romera-Paredes, H. Aung, N. Bianchi-Berthouze, and M. Pontil, "Multilinear multitask learning," in *Proc. of Int. Conf. on Mach. Learning*, 2013, pp. 1444–1452.

[25] "Ros.org," [Online]. Available: http://www.ros.org/, 2016.

[26] "Gazebo," [Online]. Available: http://gazebosim.org/, 2016.

[27] M. Aicardi, G. Casalino, A. Balestrino, and A. Bicchi, "Closed loop smooth steering of unicycle-like vehicles," in *Proc. of IEEE Conf. on Decision and Control*, 1994, pp. 2455–2458.

[28] H. Huang and J. Sturm, "tum_simulator - ros wiki," [Online]. Available: http://wiki.ros.org/tum_simulator, 2016.