Lifelong Learning Applications to Mobile Robotics

David Isele
Univ. of Pennsylvania
isele@seas.upenn.edu

José Marcio Luna
Univ. of Pennsylvania
joseluna@seas.upenn.edu

Eric Eaton
Univ. of Pennsylvania
eeaton@seas.upenn.edu

Gabriel de la Cruz
Washington State Univ.
gabriel.delacruz@wsu.edu

James Irwin
Washington State Univ.
james.irwin@wsu.edu

Brandon Kallaher
Washington State Univ.
brandon.kallaher@wsu.edu

Matthew E. Taylor
Washington State Univ.
taylorm@eecs.wsu.edu

Marcio's comment: We need to decide a good title.

ABSTRACT

Learning controllers for multiple systems is often an expensive process when controllers for each system are learned individually. Advances in lifelong learning suggest that information between systems can be shared, improving the quality of the controllers that are learned. However these results have been largely theoretical, with applications limited to benchmark problems with known dynamics. We show that these methods can be extended to robotic platforms. Particularly we validate our assumptions for transfer learning between tasks with unknown dynamics in order to carry out a disturbance rejection problem. We view this as early work leading up to learning robust fault tolerant control.

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

1. INTRODUCTION

In control systems, a perfect model of the dynamics of the system is often necessary to guarantee the stabilization of the system. This can be problematic in complicated systems where the dynamics are difficult to model or require information that is not available to the designer.

Policy search approaches have been proposed to deal with the design of controllers in model-free applications. However, it is difficult to make claims of robustness or generalization of the learned controllers

Our goal is to learn fault tolerant control in multi-agent systems. In order to approach this problem we begin by developing a method that can accomplish disturbance rejection across multiple robots. Given a collection of robots each with a different unknown disturbance, learning a unique control policy for each robot in the system can be very costly. One approach to reduce the amount of learning is to share information between robots.

Lifelong learning [19] is a promising approach for accomplishing information sharing between different robots. It works on-line, allowing the different systems to be encountered consecutively so a priori knowledge of all the different systems is not required at the start of training. Also it preserves and possibly improves the models encountered early on, in contrast to transfer methods which only optimize performance on the target system.

However most recent work in lifelong learning [19, 5, 6] has been theoretical, using simulations of benchmark problems with known dynamics to demonstrate knowledge sharing. The contribution of this work is to present the first results of lifelong learning on robotic systems. We do this by applying the PG-ELLA framework [5] to the problem of disturbance rejection on a set of Turtlebots in Gazebo Simulator. In this work in progress, the obtained results motivate future experiments on the real robot.

2 3 4 [☛ **Marcio:** The introduction needs more work, please don't forget to clarify our main contributions/goals and please provide the organization of the paper.]

2. RELATED WORK

Building mathematical models that describe the behavior of physical systems is common practice to analyze, predict and control their behavior to fulfill specific goals. Among the well known techniques for modeling physical systems we find partial, ordinary differential and difference equations [9, 17], and Discrete Event Systems (DES) such as queueing and Petri networks [7, 16, 22]. Control systems theory uses differential and difference equations to model the mechanics of physical systems. In control systems a validated model provides ways of assessing the stability and stabilizability of the system to be able to control its behavior.

Some of the typical problems in control systems are regulation, trajectory tracking, disturbance rejection and robustness among many others [9, 15, 17]. All these problems are associated with the analysis of the stabilizability of the system, as well as the design of controllers to stabilize it. These controllers consist of theoretical artifacts that would allow the solution of control problems such as the aforementioned ones.

The disturbance rejection problem consists of implement-

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.*

ing a controller that allows the plant to fulfill the desired task while compensating for a disturbance that modifies its nominal dynamics. As long as there is an accurate model of the plant, several mathematical artifacts have been provided to handle constant, constant and unknown, time-varying and even stochastic disturbances [8, 9, 15]. However, things get more complicated when no model is provided, even for simple disturbances. In a model-free setting, the goal is to design a controller that stabilizes a system whose model is not available due to complex internal iterations, uncertainty in the system, event-based dynamics and technological limitations. Unlike typical control system theory where policies are generated from a model, we generate policies based on sampled trajectories generated by simulation.

Reinforcement learning [11] is often utilized to learn controllers in a model-free settings. Amongst reinforcement learning algorithms, policy gradient (PG) methods [20, 23] are popular in robotic applications [13, 18] since they accommodate continuous state/action spaces and can scale well to high dimensional spaces. Different from these works we use policy gradients in a lifelong learning setting.

It has been shown that PG methods can be used in a lifelong learning setting [5, 6], however these works focus largely on theory, using benchmark simulations to demonstrate their results. While there are examples of lifelong learning on robots, they tend to focus in skill refinement on a single robot [10, 21] rather than sharing information across multiple robots as we do in our work.

3. BACKGROUND

Our approach works by sharing knowledge between different robotic systems. The policy for each system is learned by reinforcement learning. In this section we cover the mathematical framework that supports our experiments on lifelong learning.

3.1 Reinforcement Learning

A reinforcement learning (RL) agent must select sequential actions to maximize its expected return. RL approaches do not require previous knowledge of the system dynamics, instead, the control policies are learned 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 RL is to find the optimal policy π^* that maximizes the expected reward.

We use a class of RL algorithms known as Policy Gradient (PG) methods [20], which are particularly well suited for solving high dimensional problems with continuous state and action spaces, such as robotic control [18].

The goal of PG is to use gradient steps to optimize the expected average return:

$$\mathcal{J}(\boldsymbol{\theta}) = \int_{\mathbb{T}} p_{\boldsymbol{\theta}}(\tau)\mathcal{R}(\tau)\mathrm{d}\tau, \tag{1}$$

where $\mathbb{T}$ is the set of all trajectories and $\mathcal{R}(\tau)$ is the average per-step reward, specifically:

$$\begin{aligned} p_{\boldsymbol{\theta}} &= \prod_{h=0}^{H} p(\mathbf{x}_{h+1}|\mathbf{x}_h, \mathbf{a}_h)\pi(\mathbf{a}_h, \mathbf{x}_h) \ , \\ \mathcal{R}(\tau) &= \frac{1}{H}\sum_{h=0}^{H} r(\mathbf{s}_h, \mathbf{a}_h, \mathbf{s}_{h+1}) \ . \end{aligned}$$

Most PG methods (e.g. episodic REINFORCE [23], Natural Actor Critic [18], and PoWER [13]) 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 (FD) [12] which optimizes the return directly.

3.2 Finite Differences for Policy Search

The local optimization around an existing policy π parameterized by a parameter matrix $\boldsymbol{\theta}$ is carried out by computing changes in the policy parameters $\boldsymbol{\Delta\theta}$ that will increase the expected reward, thus producing the iterative update,

$$\boldsymbol{\theta}_{m+1} = \boldsymbol{\theta}_m + \boldsymbol{\Delta\theta}_m.$$

Gradient-based methods for policy updates follow the gradient of the expected return $\mathcal{J}$ given a step-size δ,

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

In FD gradients, we have a set of n perturbed policy parameters which are used to estimate the effect of a change in policy parameters:

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

where $\boldsymbol{\Delta\theta}_{\boldsymbol{p}}$ are the individual perturbations for $\mathbf{p} = [1, \ldots, n]$, $\Delta\hat{\mathcal{J}}_{\mathbf{p}}$ is the estimate of their effect on the return, and the $\mathcal{J}_{ref}$ is a reference return which is usually taken as the return of the unperturbed parameters. By using linear regression we get an approximation of the gradient,

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

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 $\boldsymbol{\Delta\theta}_{\boldsymbol{p}}$. 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.

In our experiments, the policy is represented as a function defined over a parameter matrix $\boldsymbol{\theta} \in \mathbb{R}^{d_\theta}$. Our goal is to optimize the expected average return of Equation 1.

In order to share information across different learned policies, we incorporate the PG learning process into a lifelong machine learning setting.

3.3 Lifelong Machine Learning

Lifelong learning focuses on learning a set of tasks consecutively while performing well across all tasks. Given a round $t = 1, \ldots, T$ a task $Z^{(t)}$ is observed. In our setting, each task corresponds to a reinforcement learning problem for an individual robot. We assume that the model associated to $Z^{(t)}$ is parameterized by a parameter $\boldsymbol{\theta}^{(t)} \in \mathbb{R}^{d_{\theta_t}}$. The ideal goal is that prior knowledge about tasks $Z^{(1)}, \ldots, Z^{(t-1)}$ should provide enough information so that the lifelong learning algorithm performs better and faster on $Z^{(t)}$ while being able to scale as the number of tasks increases.

Following work in both multi-task [14] and lifelong learning [19], we assume there is a shared basis $\boldsymbol{L} \in \mathbb{R}^{d_x \times k}$ and a sparse weight vector $\boldsymbol{s}^{(t)} \in \mathbb{R}$, so that the policy parameters $\boldsymbol{\theta}^{(t)}$ are given by,

$$\boldsymbol{\theta}^{(t)} = \boldsymbol{L}\boldsymbol{s}^{(t)}.$$

Using the return function in (1) we propose the following multi-task objective function,

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

where $\boldsymbol{S}$ is the set of the sparse vectors $\boldsymbol{s}^{(t)}$, the L1 norm of $\|\boldsymbol{s}^{(t)}\|_1$ provides sparse code for $\boldsymbol{s}^{(t)}$ and the Frobenious norm in $\|\boldsymbol{L}\|_F^2$ provides regularization of $\mathbf{L}$. The coefficients μ and $\lambda \in \mathbb{R}$ are weights for the regularization and sparsity respectively. The learning objective function is approximated by a second order Taylor expansion around an estimate $\boldsymbol{\alpha}^{(t)}$ of the single task policy parameters of task t. The optimization problem is solved by using the on-line ELLA algorithm introduced in [19] and extended to reinforcement learning in [5]. The optimization problem is solved by incrementally updating $\mathbf{L}$ by the following the update equations,

$$\begin{aligned} \boldsymbol{s}^{(t)} &\leftarrow \arg\min_{\boldsymbol{s}} \left\| \boldsymbol{\alpha}^{(t)} - \boldsymbol{L}\boldsymbol{s}^{(t)} \right\|_{\boldsymbol{\Gamma}^{(t)}}^2 + \mu \|\boldsymbol{s}\|_1 \quad , \\ A &\leftarrow A + \left(\boldsymbol{s}^{(t)} \boldsymbol{s}^{(t)\mathsf{T}} \right) \otimes \boldsymbol{\Gamma}^{(t)} \quad , \\ b &\leftarrow b + \mathrm{vec}\left(\boldsymbol{s}^{(t)} \otimes \left(\boldsymbol{\theta}^{(t)\mathsf{T}} \boldsymbol{\Gamma}^{(t)} \right) \right) \quad , \\ L &\leftarrow \mathrm{mat}\left(\left(\tfrac{1}{T}\boldsymbol{A} + \lambda \boldsymbol{I}_{l \times d_\theta, l \times d_\theta} \right)^{-1} \tfrac{1}{T}\boldsymbol{b} \right) \quad . \end{aligned} \tag{2}$$

where $\|\boldsymbol{v}\|_{\boldsymbol{A}}^2 = \boldsymbol{v}^\mathsf{T}\boldsymbol{A}\boldsymbol{v}$ and $\boldsymbol{\Gamma}^{(t)}$ is the Hessian of the PG objective function, and $\boldsymbol{A}$ and $\boldsymbol{b}$ are initialized to zero matrices. The full algorithm is described in Algorithm 1

Algorithm 1 PG-ELLA (k, λ, μ)

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

4. EXPERIMENTS

We evaluated our approach by modeling the control policies for different Turtlebot systems [3]. Turtlebots are an affordable robotic platform that uses the Robotic Operating System (ROS) in its hydro version, which is compatible with the Gazebo simulator [1, 2]. We artificially induce a random and constant disturbance to the control signal to emulate a bias on the angular velocity of each robot. Each turtlebot has a different constant disturbance drawn uniformly from $[-0.1, 0.1] \subset \mathbb{R}$ and measured in m/s. These limits were selected to provide a large noise that was still within the bounds of what the turtlebot correct.

The Gazebo simulator considers the kinematics and mechanics of the system. In a first attempt to explore disturbance rejection as a preamble of fault tolerant control applications, the Turtlebot should learn how to drive itself from an initial to a goal point. It should accomplish this simple task while being affected by a constant and unknown angular disturbance. Thus the robot is enforced to compensate for the induced failure. It is worth mentioning that the difficulty of the disturbance will be increased by assuming time varying disturbance and stochastic disturbances later on.

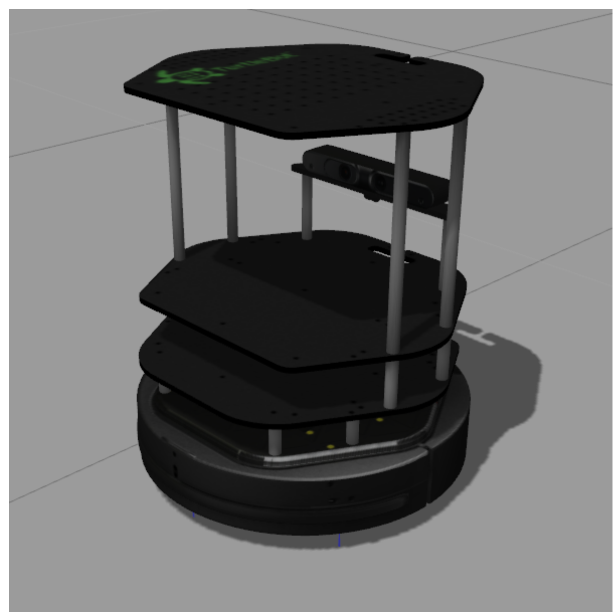

Figure 1: Turtlebot model in Gazebo Simulator.

We assume we have little knowledge of the Turtlebot model, so we use the simplified kinematic model provided in [4]. In the kinematic model, the state space is given by $\boldsymbol{x} \in \mathbb{R}^3$ and the action space is described by $\boldsymbol{a} \in \mathbb{R}^2$. Notice that the model in [4] just considers the kinematics of a unicycle in polar coordinates so we do not consider the dynamics of the system, *i.e.,* we are neglecting model parameters such as mass, damping and friction coefficients, as well as inputs such as forces and torques. Then, the nonlinear policy is derived neglecting the dynamics and just assuming simple kinematics, therefore, taking into account that our action is given by $\boldsymbol{a} = \boldsymbol{\theta}^\mathsf{T}\boldsymbol{\phi}(\boldsymbol{x}) = (u, w)^\mathsf{T}$ where $\boldsymbol{x} = (\rho, \gamma, \psi)^\mathsf{T}$ as illustrated in Fig. 3 we propose the following nonlinear gain vector structure,

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{pmatrix} \rho\cos(\gamma) \\ \gamma \\ \frac{\cos(\gamma)\sin(\gamma)}{\gamma}(\gamma+\psi) \\ 1 \end{pmatrix} \tag{3}$$

where ρ, γ and ψ are indicated in Fig. 2.

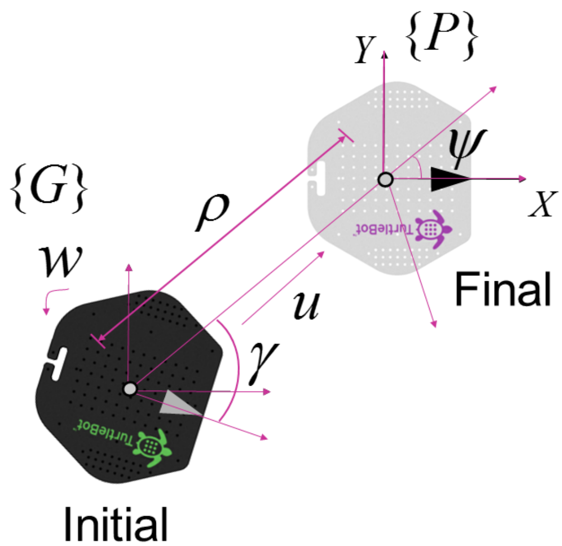

Figure 2: State variables of simplified go-to-goal problem.

The state is a non-linear transformation of the position and heading angle. This implementation was entirely done in Python. We ran experiments using natural actor critic [18], episodic REINFORCE [23], and FD [12] and found FD to work best for our problem. We believe this is a result of the particular nonlinear transformation we are using and the fact that FD optimizes the true return. In future work we will present a comparison of different base learners.

4.1 Methodology

We start by generating 20 robots, each with a different constant disturbance and a unique goal, both selected uniformly. We use 20 robots because it provdes a large diversity in tasks but is a number that is still practical to run in simulation. The robots are run for 20 learning steps of FD, each learning step consists of 15 roll-outs of 50 time steps each. Note that 20 learning steps is fewer iterations than is required for any system to converge to a good controller. The number of roll-outs and time steps were selected to allow for successful learning while minimizing the runtime. The policy that is learned after 20 iterations of FD is used as θ^* for PG-ELLA. We use a learning rate of 1×10^{-6}.

We learn our PG-ELLA knowledge repository $\boldsymbol{L}$ and sparse representation $\boldsymbol{s}^{(t)}$ using the update equations given by (2). Tasks are encountered randomly with repetition and learning stops once every task has been seen once. For our experiments we approximate the hessian with the identity matrix. We select $k = 8$ to be the number of columns, and use sparsity coefficient $\mu = 1 \times 10^{-3}$ and regularization coefficient $\lambda = 1 \times 10^{-8}$.These coefficients were selected by trial and error.

We then compare the policy reconstructed from PG-ELLA against the policy that was learned after 20 iterations of FD by comparing the learning curves that result from running FD for an additional 80 learning iterations. Performance is averaged over 6 trials for all 20 robots to increase our confidence in the results. In Figure 3 we see that PG-ELLA is successfully able to reconstruct the policies and provide an additional benefit of positive transfer. We plot the change in reward of using PG-ELLA instead of PG in Figure 4, and observe that PG-ELLA improves the quality of the learned controllers. Note that these are preliminary results and we suspect further refinements will enable us to achieve larger amounts of transfer.

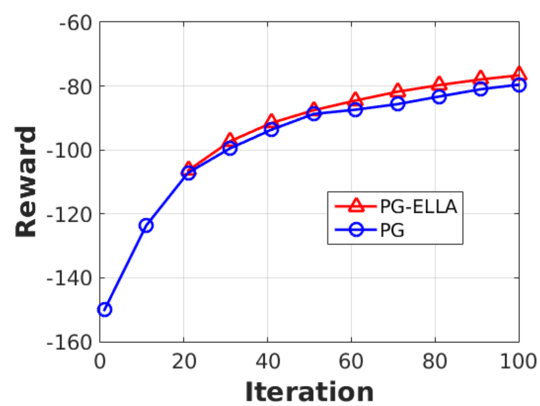

Figure 3: Learning curves for FD and PG-ELLA using FD to transfer information between tasks.

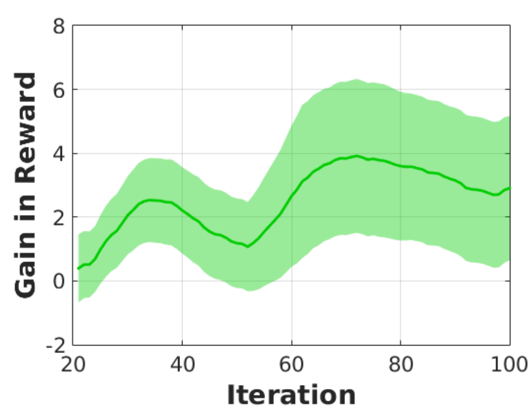

Figure 4: The positive transfer achieved by using ELLA.

5. CONCLUSIONS

We demonstrate the use of lifelong learning for disturbance rejection on Turtlebots. This is intended to lay the foundation for creating fault tolerant control on multi-agent systems. We show that PG-ELLA can be implemented on simulated turtlebots and that it outperforms standard PG. This suggests that PG-ELLA can be extended to real robotic systems.

Acknowledgments

the grant goes here

6. ADDITIONAL AUTHORS

REFERENCES

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

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

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

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

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

[6] H. Bou Ammar, E. Eaton, P. Ruvolo, and 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 and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, New York, 2nd edition, 2008.

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

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

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

[11] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, page 0278364913495721, 2013.

[12] J. Kober, J. A. D. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, July 2013.

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

[14] A. Kumar and H. Daume III. Learning task grouping and overlap in multi-task learning. *arXiv preprint arXiv:1206.6417*, 2012.

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

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

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

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

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

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

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

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

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