Generalizing Policy Advice with Gaussian Process Bandits for Dynamic Skill Improvement

Jared Glover and **Charlotte Zhu**
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Email: {jglov,charz}@mit.edu

Abstract

We present a ping-pong-playing robot that learns to improve its swings with human advice. Our method learns a reward function over the joint space of task and policy parameters $\mathcal{T} \times \mathcal{P}$, so the robot can explore policy space more intelligently in a way that trades off exploration vs. exploitation to maximize the total cumulative reward over time. Multimodal stochastic polices can also easily be learned with this approach when the reward function is multimodal in the policy parameters. We extend the recently-developed Gaussian Process Bandit Optimization framework to include exploration-bias advice from human domain experts, using a novel algorithm called Exploration Bias with Directional Advice (EBDA).

Introduction

As any athlete (or parent) knows, coaching is an essential ingredient to human motor skill learning for manipulation. Learning to tie one's shoes, cook a meal, or shoot a basketball is nearly impossible without the active guidance of a teacher. With the right combination of demonstration, verbal advice, goal-setting, and performance feedback, humans routinely teach one another incredibly complex skills in a wide variety of domains. Yet most work on manipulation learning in robotics has focused on self-improvement. The human teacher may be involved in an initial demonstration via teleoperation or kinesthetic teaching, but after that the robot is on its own; left to explore independently, often using no more than a random walk in policy space, with rewards few and far between.

The work that has been done on coaching robots (beyond the initial demonstration phase) has primarily used goal-setting and feedback (i.e., rewards). But there has been very little work on directly giving robots policy exploration advice. Sometimes a simple "pull harder" or "keep your knees bent" is enough to get human learners to master a skill. Doing the same for robots in arbitrary domains will require AI systems that can translate generic natural language commands into motor skill policy information. While there has recently been some effort on this problem (Tellex et al. 2011), we are a long way from any complete solution.

In this work we assume that the mapping between verbal commands and policy parameters can be programmed in ahead of time. Our focus is on integrating an advice signal from a human coach into a robot's exploration policy. We focus on episodic learning tasks, like hitting a ping pong ball, where the robot executes a policy π_θ with parameters θ for some finite period of time, then receives a reward y reflecting how well it performed the task. In the case of ping pong, the policy π_θ is a computer program (parameterized by θ) for swinging a robot arm, and the reward depends on whether or not the robot succesfully hit the ball back to the other side of the table. The robot's goal is to maximize the total reward it receives over time. We phrase this as a multi-armed bandit problem, and extend the recently-developed Gaussian Process Bandit Optimization framework (Srinivas et al. 2009; Krause and Ong 2011; Contal and Vayatis 2013) to include advice in the regret-minimizing exploration/exploitation strategy.

Policy Search for Dynamic Skill Improvement

There is a long-running debate in robotics on whether model-based or model-free learning of motor control policies is the best way to get robots to perform dynamic tasks like walking, juggling, or playing ping pong. Proponents of model-based approaches boast that with carefully-tuned models of system dynamics, robots can achieve skills equalling or even surpassing those of their human teachers, while the model-free crowd argues that acheiving such models isn't always possible, and that robots' ability to learn new tasks from demonstration, with little or no prior domain-specific programming, is the true measure of success.

Regardless of whether a robot uses model-based or model-free methods to achieve basic competency at a new skill, skill improvement for model-based methods is limited by the accuracy of the model class. In contrast, model-free methods, which directly explore the space of motor control policies, are only limited by the class of policies. Of course, this space is often too large to search exhaustively, and aggressive search heuristics and priors are needed to increase the data efficiency of model-free learning algorithms in all but the simplest domains.

Policies must also typically be generalized over a task space $\mathcal{T}$, which we here parameterize by a task parameter vector τ. For example, in our application τ includes the posi-

tion, velocity and spin of the incoming ball when it bounces on the robot's side of the table. Complicating matters is that τ is usually a hidden variable: the robot is given a noisy task observation, $\hat{\tau}$, from which it must infer the underlying task parameters.

Representing the policy: explicit vs. implicit

There are two main ways that a policy search algorithm can represent policies—explicitly, with a direct mapping from task parameters to policy parameters ($\mathcal{T} \rightarrow \mathcal{P}$); or implicitly, with a reward function over the joint space of task and policy parameters ($\mathcal{T} \times \mathcal{P} \rightarrow \mathbb{R}$)[1]. Although both representations have been tried in classic reinforcement learning domains (Engel, Mannor, and Meir 2005), only direct policy mappings have been used in recent work where robots have learned dynamic manipulation skills, like throwing darts, flipping pancakes, or hitting balls (Kober et al. 2012; Kormushev, Calinon, and Caldwell 2010; Da Silva, Konidaris, and Barto 2012). In this (direct) approach, a continuous (Kober et al. 2012) or piecewise-continuous (Da Silva, Konidaris, and Barto 2012) function Φ is learned from $\mathcal{T}$ to $\mathcal{P}$, representing the robot's current estimate of the reward-maximizing policy parameters for each task parameter vector τ. To explore, the robot perturbs Φ with random noise; for example, by sampling from a Gaussian process (Kober et al. 2012).

There are two problems with this direct-mapping-based exploration policy. First, it may be difficult for the learning algorithm to escape from local maxima, since the exploration only makes local perturbations to the policy parameters. Second, it restricts the class of stochastic policies that an agent can learn to those that are unimodal in policy parameter space.

In contrast, by learning a reward function over the joint space of task and policy parameters $\mathcal{T} \times \mathcal{P}$, the robot can explore policy space more intelligently in a way that trades off exploration vs. exploitation to maximize the total cumulative reward over time. Multimodal stochastic polices can also easily be learned with this approach when the reward function is multimodal in the policy parameters. In figure 1, we illustrate the difference between implicit and explicit policy representations. For smooth reward functions that are unimodal in $\mathcal{P}$ (figure 1, left), both explicit and implicit policy models are suitable. But for "bumpy" reward functions with many local maxima, explicit methods will lose information (figure 1, right). Note that for many dynamic robotic tasks like ping pong, both $\mathcal{T}$ and $\mathcal{P}$ are continuous.

Gaussian Process Bandit Optimization

The multi-armed bandit problem (Robbins 1952) is the decision problem of choosing the sequence of "levers" to pull when each lever ℓ has an associated "payoff" or reward distribution, D_ℓ. The agent pulls a lever, receives a reward, and then gets to choose again, incorporating all the information it has obtained from past lever pulls.

[1]The learning domains in this work are episodic (not sequential).

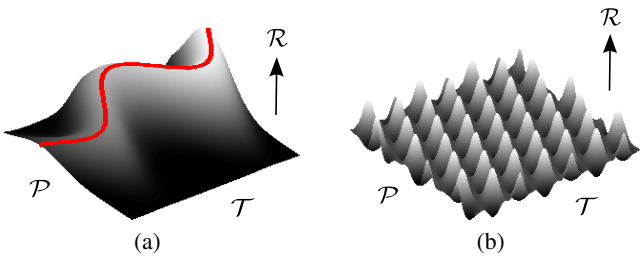


Figure 1: Implicit vs. explicit policies. In the smooth reward function on the left, a good explicit policy mapping $\mathcal{T}$ to $\mathcal{P}$ can be found. In the reward function on the right, any explicit policy mapping will be throwing away information about alternative modes.

Maximizing the reward that a robot learner gets by exploring a reward function $f : \mathcal{T} \times \mathcal{P} \rightarrow \mathbb{R}$ mapping continuous task and policy parameters to the reals is akin to a multi-armed bandit problem with infinitely many levers. At each round t, the world chooses task parameters τ_t which the robot observes with noise as $\hat{\tau}_t$, then the robot selects policy parameters θ_t, executes policy π_{θ_t}, and receives a noisy reward $y_t = f(\tau_t, \theta_t) + \epsilon_t$.

With some mild smoothness assumptions, the reward function f can be modelled as a Gaussian process (GP). There is a large body of literature on exploration heuristics for GPs (Mockus 1989; Brochu, Cora, and De Freitas 2010). We use GP-UCB (Srinivas et al. 2009; Krause and Ong 2011), which has recently been shown to achieve sublinear regret bounds for cumulative reward in bounded policy domains $\mathcal{P}$. It has not yet been applied to active learning of robot control parameters.

GP-UCB (Gaussian Process Upper Confidence Bound) The GP-UCB algorithm (Srinivas et al. 2009; Krause and Ong 2011) chooses the next θ_t so as to maximize a combination of the current GP mean and variance estimates of the reward function f:

$$\theta_t = \arg\max_{\theta} \mu_{t-1}(\hat{\tau}_t, \theta) + \beta_t^{1/2} \sigma_{t-1}(\hat{\tau}_t, \theta), \qquad (1)$$

where $\mu(\cdot)$ and $\sigma(\cdot)$ are the posterior mean and standard deviations functions of the GP after the first $t-1$ observations, $(\hat{\tau}_1, \theta_1, y_1), \ldots, (\hat{\tau}_{t-1}, \theta_{t-1}, y_{t-1})$. The β_t's are exploration constants, which must increase over time as a function of $\log t$ in order to obtain the theoretical regret bounds below.

Letting θ_t^* be the reward-maximizing policy parameter vector for the true task parameters τ_t, the *true cumulative regret* up to time step T is defined as

$$R_T := \sum_{t=1}^{T} f(\tau_t, \theta_t^*) - f(\tau_t, \theta_t). \qquad (2)$$

Unfortunately, since the τ_t's are unobserved, the best bound we can achieve for R_T is $\mathcal{O}(T)$, since there will always be a small constant expected regret at every time step t due to task parameter observation noise, $\tau_t - \hat{\tau}_t$.

The sub-linear regret bounds that have been found for GP-UCB assume that the agent has access to the true task parameters. We can achieve similar bounds for the noisy task parameter case with a modified definition of regret. Letting $\hat{\theta}_t^*$ be the reward-maximizing policy parameter vector for task parameters $\hat{\tau}_t$, we define the *observable cumulative regret* up to time step T as

$$\hat{R_T} := \sum_{t=1}^{T} f(\hat{\tau}_t, \hat{\theta}_t^*) - f(\hat{\tau}_t, \theta_t). \quad (3)$$

$\hat{R_T}$ has a bound of $\mathcal{O}(\sqrt{T}\beta_T\gamma_T)$ with probability $1-\delta$, where $\delta \in (0,1)$ is an exploration parameter that can be controlled by the user. Since the robot doesn't get to choose τ_t, it is unfair to penalize it for the difference between τ_t and $\hat{\tau}_t$, and so the observable regret bounds still provide some insight into our problem.

Giving Advice

There are many ways to train a robot. Some of the most common are:

- **Demonstration:** The coach can demonstrate a particular skill to a robot, either by performing the skill while the robot watches, or by teleoperating the robot.
- **Goal setting:** The coach can select a sequence of tasks to give the robot, only moving on when the robot has demonstrated a sufficient level of skill at the current task.
- **Performance feedback:** The coach can give the robot learner a reward signal, which may be in addition to environmental rewards that the robot observes on its own.
- **Policy advice:** The coach can suggest modifications to the robot's policy parameters, like *"hit the ball harder"* or *"angle your paddle down more"*.

Learning from demonstration is a well-explored paradigm in robotics (Atkeson and Schaal 1997; Peters et al. 2011). However, it can be difficult for a human to teleoperate a robot to perform complex and dynamic tasks, and computer vision algorithms are not yet advanced enough to understand most manipulation actions (for example, to determine that a human is using their right hand to hold a jar between their knees while the left hand tries to generate enough force to twist off the cap). Nevertheless, learning from demonstration has been applied successfully to several robotic domains, including ping pong (Mülling et al. 2013).

Setting goals is less explored in the robotics literature, but it is the *de facto* standard for all of robotics research. Typically, a human researcher explores policy parameter space (by modifying the robot's program) rather than the robot exploring on its own.

Performance feedback is perhaps the most-explored mechanism for training computer agents (not just robots). Particularly for reinforcement learning domains (MDPs) like computer games and robot navigation, there is a long history of incorporating a human reward signal (called *reward shaping*) into reinforcement learning algorithms (Griffith et al. 2013; Thomaz and Breazeal 2008; Judah et al. 2010; Pilarski and Sutton 2012).

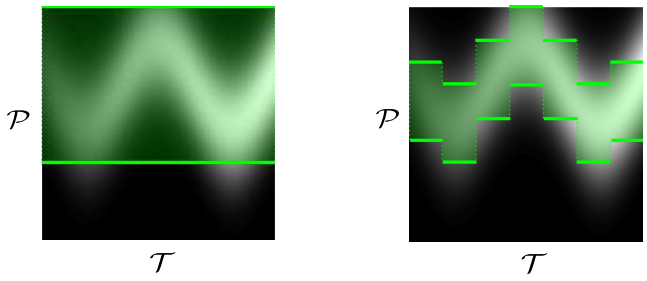


Figure 2: The robot interprets advice from a human coach as an exploration bias. The light green shaded areas on top of the reward functions (top-down view of figure 1(a)) in the left and right images indicate what the robots believes are "good" regions to explore, and are inferred from a series of policy gradients given by the human coach (e.g., "swing harder" or "angle your paddle up more"). Note that the exploration advice can either be global (left) or local (right). In our system, local advice is the default; advice is only applied globally (for all $\tau \in \mathcal{T}$) at the coach's discretion (e.g., by saying "*always* swing harder").

The approach to training robots that we use in this paper is that of giving the robot policy advice. In RL domains, this has been explored with *action biasing*–where the human biases the robot's action-selection mechanism (Knox and Stone 2012), and *control sharing*–where the human selects some of the robot's actions directly (Smart and Kaelbling 2002). For manipulation, a recent technique is to give the robot *policy improvement* advice–where the human provides the robot with incremental improvements to its manipulation trajectories in order to indicate a step in policy space that may increase reward (Jain, Joachims, and Saxena 2013).

Policy gradient advice

Giving robots entirely new policies as advice can be a time-consuming process. Instead, we explore *policy gradient* advice, where the human trainer need only provide a direction in policy space $\mathcal{P}$ in which reward is likely to increase, rather than a new, complete policy. Intuitively, this type of advice can directly capture information like *"hit the ball harder"* or *"angle your paddle down more"*, so long as the mapping between the natural language instructions and policy parameters is known.

There are two ways a robot can interpret policy gradient advice: (1) as a **pseudo-reward**, or (2) as an **exploration bias**. Treating the advice as a pseudo-reward involves a level of trust: the robot believes that hitting the ball harder is a good thing to do because the coach said so, and it may take several trials to overcome the pseudo-reward in the case that the coach is wrong. Using the policy gradient advice as an exploration bias (which is akin to *action biasing* in RL domains) is less susceptible to bad advice, since the robot can decide for itself whether the advice was useful. In addition, gradient-based pseudo-reward may be difficult to implement in continuous policy domains, since it isn't clear how far in the direction of the gradient one should place the pseudo-

rewards, or how many to place. For these reasons, we interpret policy gradient advice as exploration bias.

Exploration Bias with Directional Advice (EBDA)

We now present our algorithm for turning directional (policy gradient) advice into an exploration bias. Intuitively, EBDA turns directional information into soft constraints on future policy parameters; namely, that they be in roughly the same direction as the advice indicates.

Let $\tau_1, \ldots, \tau_n$ be a sequence of tasks the robot has received, let $\theta_1, \ldots, \theta_n$ be the policy parameters it chose for each task, and let $\psi_1, \ldots, \psi_n$ be the coach's policy gradient advice. The direction of each ψ_i indicates a direction of increasing reward (according to the coach) in policy parameter space, with respect to the current θ_i.

The robot interprets $\psi_1, \ldots, \psi_n$ as a sequence of probabilistic constraints on the policy parameter vector θ_{n+1} for the next task, τ_{n+1}. Each constraint is of the form $\{\theta : \psi_i \cdot (\theta - \theta_i) > 0\}$, and is only applied if the advice is global (indicated by $|\psi_i| > 1$) or if the new task parameter vector τ_{n+1} is close to τ_i. EBDA then flips a biased coin (based on an advice noise parameter ρ) to determine whether or not to include each constraint. Pseudo-code for EBDA is shown in Algorithm 1.

Algorithm 1 EBDA

Input: History of task parameters $\tau_1, \ldots, \tau_n$, policy parameters $\theta_1, \ldots, \theta_n$, policy gradient advice $\psi_1, \ldots, \psi_n$, advice noise ρ, and a new task parameter vector τ_{n+1}
Output: Exploration region $\mathcal{R} \subset \mathcal{P}$

```
R ← P
for i = 1 to n do
    Sample s ~ Uniform(0, 1)
    if |ψ_i| > 1 or |τ_{n+1} − τ_i| < τ_thresh then
        if s > 1 − ρ then
            R ← R ∩ {θ : ψ_i · (θ − θ_i) > 0}
        end if
    end if
end for
```

Robot Ping Pong

There are several reasons for choosing ping pong as an application to study human coaching for dynamic robots. First, ping pong is a high-speed, underactuated control task, requiring a level of precision and timing not seen in many robot systems outside of the factory setting. At the same time there is substantial uncertainty about the world. The spin on the ball is difficult to observe directly, and the opponent's actions are unpredictable. For these reasons it is very difficult to conquer with model-based optimal control methods alone, leaving room for learning methods to improve significantly upon the state of the art. Ping pong is also similar in spirit to other high-skill manipulation tasks, such as those often found in cooking—chopping vegetables, stirring ingredients in a bowl, flipping burgers—in that they are all dynamic and underactuated and are defined by repetitive motions applied to a limited range of objects. Finally, ping pong has been used as an application for learning dynamic control policies (Mülling et al. 2013), which makes it a good application for comparison.

Figure 3: Our ping pong robot uses a 7-dof Barrett WAM arm with a ping pong paddle rigidly attached at the wrist.

Background There have been several robot ping pong systems since the late 1980s (Andersson 1988; Miyazaki, Matsushima, and Takeuchi 2006; Sun et al. 2011; Mülling et al. 2013). The closest work to ours is (Mülling et al. 2013). Their robot also learns to improve its swing parameters from experience, but they use an explicit policy mapping from task parameters to policy parameters, and they do not use human advice to help the robot learn. We compare to their learning method, CrKR, in our experimental results.

Hardware Our system consists of a 7-dof Barrett WAM arm, mounted sideways on the wall behind the ping pong table, along with a pair of high-speed 200Hz black-and-white Silicon Video SV-640M cameras mounted on a fixed frame above the arm (Figure 3). A Kinect is also used to detect human opponents and predict ball spin based on the type of swing they use to hit the ball to the robot.

Sensing Since the cameras are stationary and the ping pong ball is much lighter than the table, we use a simple background-subtraction-based blob tracker to detect and track the ping pong ball. Detections in the left and right cameras are triangulated to estimate depth, and the ball's 3-D position is then transformed into ping pong table coordinates. The camera-table transform is estimated using a gradient descent algorithm which optimizes the fitness of the table's edges projected into left and right Canny edge images. The human opponent's body configuration is tracked in the Kinect depth image with OpenNI.

Tracking and predicting the ball's trajectory In order to decide when and where the robot should swing its paddle, it needs to predict the ball's future trajectory based on its past trajectory. Although it would be possible for the robot to learn the ball's dynamics autonomously, this is not the focus of our current work. Therefore, for simplicity we use Andersson's models to predict ball aerodynamics and collisions with the table (Andersson 1988).

The ball's state is defined by the 9-dimensional vector $\mathbf{b} = (x, y, z, v_x, v_y, v_z, w_x, w_y, w_z)$, representing the ball's current position, velocity, and spin. During flight, the ball's

acceleration is given by

$$\mathbf{a} = -C_d|\mathbf{v}|\mathbf{v} + C_m|\mathbf{v}|\mathbf{w} \times \mathbf{v} - \mathbf{g}, \quad (4)$$

where C_d and C_m are drag and Magnus coefficients, and $\mathbf{g}$ is the gravity acceleration vector (in the negative z-direction).

The bounce dynamics model is slightly more complex, since the ball may change from a "slipping" to a "rolling" mode during contact with the table. First define

$$\mathbf{v_{rel}} = (v_x - w_y r, v_y + w_x r, 0) \quad (5)$$

$$\hat{\mathbf{v}}_\mathbf{r} = \frac{\mathbf{v_{rel}}}{\|\mathbf{v_{rel}}\|} \quad (6)$$

where $r = 20mm$ is the ball's radius. Then rolling occurs if

$$\|\mathbf{v_{rel}}\| < -\frac{5}{2} C_f v_z (1 + C_r) \quad (7)$$

where C_f is the coefficient of friction and C_r is the coefficient of restitution. If rolling occurs during the bounce, the ball's final velocity $\mathbf{v_f}$ and spin $\mathbf{w_f}$ are

$$\mathbf{v_f} = (\frac{3v_x + 2rw_y}{5}, \frac{3v_y - 2rw_x}{5}, -C_r v_z) \quad (8)$$

$$\mathbf{w_f} = (\frac{2rw_x - 3v_y}{5r}, \frac{2rw_y + 3v_x}{5r}, w_z). \quad (9)$$

Otherwise, the ball experiences friction during the whole bounce, and the final velocity and spin are

$$\mathbf{v_f} = \mathbf{v} + (C_f \hat{\mathbf{v}}_\mathbf{r} - \hat{\mathbf{k}}) v_z (1 + C_r) \quad (10)$$

$$\mathbf{w_f} = \mathbf{w} + \frac{3C_f}{2r} (\hat{v}_{ry}, -\hat{v}_{rx}, 0) v_z (1 + C_r) \quad (11)$$

where $\hat{k} = (0, 0, 1)$ is the table normal.

To track the ball's state, we use an Extended Kalman Filter (EKF) derived from these equations of motion. We initialize the ball's spin vector based on the human opponent's paddle hand velocity vector (determined by smoothing a finite difference of hand position signals from OpenNI) at the time of contact with the ball. Letting (h_x, h_y, h_z) be the hand's velocity vector, we empirically set the ball's spin to be $\mathbf{w} = (240, 40h_x, 0)$ if $h_z > 0$ for a topspin stroke, and $\mathbf{w} = (-240, -80h_x, 0)$ otherwise[2].

Swing planning Our approach to swing planning is hierarchical. First, the robot finds a *hit plan*—indicating a desired contact between the ball and paddle. The hit plan includes the paddle's position, velocity, normal, and time when it hits the ball. Then, a kinodynamic paddle trajectory is planned that achieves the desired contact state. Finally, the robot uses inverse kinematics to find a trajectory in joint angle space to satisfy the paddle trajectory constraints.

For our learning experiments, the hit plan is precisely the policy parameter vector, θ. Task parameters τ are the position, velocity and spin of the incoming ball when it bounces on the robot's side of the table.

[2]This is clearly a gross approximation to the true effect of the human hand's velocity vector on the initial ball spin, but with high initial EKF variance, they provide a suitable prior on the ball's spin state for our experiments.

The paddle always starts in the same configuration with position $p_0 = (x_0, y_0, z_0)$ and normal $n_0 = (n_{x_0}, n_{y_0}, n_{z_0})$ at the beginning of every swing. To find a paddle trajectory that achieves the desired hit (paddle position p_{hit}, normal n_{hit}, and velocity v_{hit} at time t_{hit}), we compute three separate *bang-bang* control laws for the paddle's x–, y–, and z–trajectories, and we interpolate the paddle normals to transition smoothly from n_0 to n_{hit}.

Let T be the duration of the swing (starting at $t_0 = t_{hit} - T$), discretized into N time steps (so each time step is $\Delta t = T/N$). The *bang-bang* control law we use is to accelerate at $-a$ for k time steps, then at $+a$ for $N - k$ time steps. Assuming the system starts at rest at the origin, the velocity trajectory given by this control law is $\Delta t \cdot a \cdot (-1, \ldots, -k, -k+1, \ldots, N-2k)$, and so the final position is

$$x_N = (\Delta t)^2 a \Big(\frac{(N-2k)(N-2k+1)}{2} - k^2 \Big). \quad (12)$$

Thus the acceleration is uniquely determined by x_N and k; we set a to make $x_N = x_{hit}$, so the paddle always reaches the desired final position. Changing k yields different final velocities, $v_N = \Delta t \cdot a \cdot (N - 2k)$; we choose the k that makes v_N as close as possible to the desired hit velocity. Note that we compute three different pairs: (a_x, k_x), (a_y, k_y), (a_z, k_z) for controllers in the paddle's x, y, and z coordinates.

Learning a swing controller Given a planned trajectory $Q = (q_0, \ldots, q_N)$ from times t_0 to t_N in joint angle space, the robot must find a way to execute the swing by sending torque commands to the motors at each joint. This is accomplished with feed-forward + proportional control in joint angle space. Given current joint angles and velocities q and $\dot{q}$, and desired joint angles and velocities q_t and $\dot{q}_t$, the robot applies torques:

$$u = K_p(q_t - q) + K_d(\dot{q}_t - \dot{q}) + u_t, \quad (13)$$

where K_p and K_d are gain matrices, and u_t are the feed-forward (constant) torques at time t.

Controllers of this form have been applied to execute dynamic motions with robot arms in the past. Often, u_t is computed with an inverse dynamics model that estimates the torques needed to achieve varying joint accelerations $\ddot{q}$ in state $(q, \dot{q})$. Such models assume $\ddot{q}$ is a function of q, $\dot{q}$, and u; higher-order effects like vibrations, cable stretching, gear thrashing, etc. are ignored.

Our ping pong robot needs to operate at the very limits of its specifications, with paddle speeds up to 3 meters/second and swing durations of less than half a second. Because of this, we found inverse dynamics models to be too noisy to rely on for swing control. Instead, our system learns a library of feed-forward torque trajectories, $U_1, \ldots, U_m$, corresponding to hit plans $H_1, \ldots, H_m$ (which uniquely determine joint trajectories $Q_1, \ldots, Q_m$. This library is learned incrementally–a new (U, H) pair is only added when the error between a desired swing and actual swing (as measured by joint encoders) is larger than a given threshold.

To learn a new swing (U, H) with corresponding Q and $\dot{Q}$, the robot initializes U to either: (i) the U_i corresponding

to the closest H_i if $|H - H_i| <$ thresh, or (ii) $G(Q_i)$: the torques needed to cancel out gravity at each joint angle configuration in Q_i[3]. Then, the robot executes the current swing according to equation 13 and computes the errors between desired and measured joint angles and velocities, $Q - \hat{Q}$ and $\dot{Q} - \hat{\dot{Q}}$. These errors are used to update U:

$$dU \leftarrow \lambda K_p(Q - \hat{Q}) + K_d(\dot{Q} - \hat{\dot{Q}}) \quad (14)$$

$$U \leftarrow U + \min(\max(-dU_{max}, dU), dU_{max}), \quad (15)$$

and then the robot repeats the swing and updates U until convergence. This U update is a functional control law for trajectories that tries to drive $Q - \hat{Q}$ and $\dot{Q} - \hat{\dot{Q}}$ to zero over time as the robot practices the swing over and over again. Note that each time the robot practices the swing, it decreases the position gains K_p in equation 13 so it relies more and more heavily on the feed-forward torques U to drive the swing.

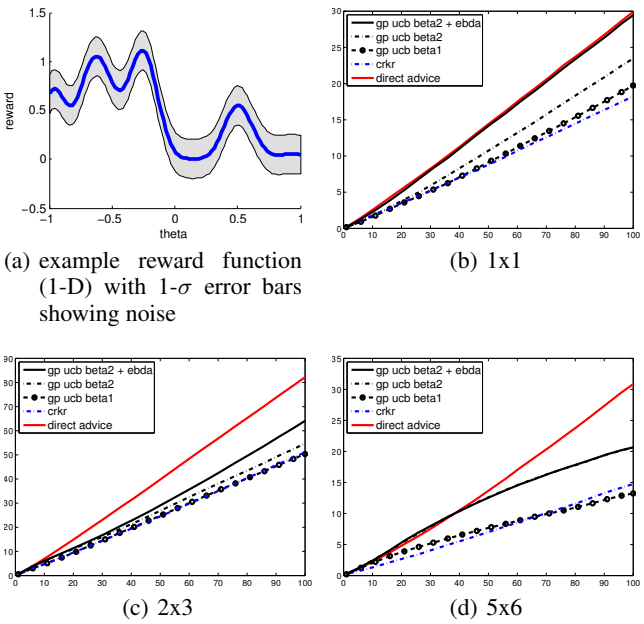
Experimental Results

We compare five learning methods: **GP-UCB**, **CrKR**, **GP-UCB + EBDA**, **CrKR + EBDA**, and **Direct**—where the human coach directly controls the robot's hit plan with incremental changes to policy parameters. CrKR is a direct policy mapping method (Kober et al. 2012) that has been applied to learning ping pong swings (Mülling et al. 2013). For the three learning algorithms that use advice, we only allowed the coach to give information about one parameter at a time.

Simulation As a pilot domain, we simulated mixture-of-Gaussian reward functions and ran each learning method (except CrKR + EBDA) for 100 trials. Direct advice was given by adding random noise to the true maximum policy parameter θ^* in the slice of $\mathcal{P}$ with all other policy parameters fixed to their last values. Directional advice ψ for EBDA was given by the direction of θ^* with respect to the last θ along one randomly chosen axis. We repeated each experiment 50 times. Average cumulative rewards for each method are shown in figure 4. For GP-UCB, we tried using both the β from the original paper (Srinivas et al. 2009) and $\beta = \log t$ (beta1 and beta2 in figure 4). Since $\beta = \log t$ performed the best in simulation, we used that beta throughout the rest of the experiments. Adding advice (with EBDA) improved GP-UCB further, while direct advice performed the best (because it is an oracle + noise).

Robot Ping Pong We next compared the learning algorithms on the task of improving the ping pong robot's hit plans. We ran each algorithm for 100 trials of a human ping pong expert hitting balls to the robot in a predetermined sequence (e.g., "high backhand topspin to (.2,.4) on the table", "low forehand underspin to (.7,.5) on the table") . If an incoming ball trajectory from the human differed significantly from the desired ball trajectory, the human was prompted to hit another ball to the robot.

[3]The anti-gravity torques are also learned: the robot moves to a list of pre-specified configurations with a PID control law and records the torques needed to keep the system at rest.



(a) example reward function (1-D) with 1-σ error bars showing noise

(b) 1x1

(c) 2x3

(d) 5x6

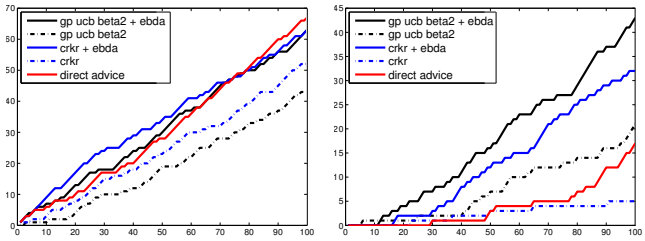
Figure 4: Simulation results (cumulative reward vs. learning trials) with mixture-of-Gaussian reward functions, for task and parameter spaces of different dimensions.

As a starting point and a baseline, we implemented a default hit plan as follows. Given an estimate of the ball's predicted future trajectory, it finds the ball's highest point in that trajectory within $\pm 15cm$ of the robot's end of the table. That ball position is the hit plan's paddle position. The paddle velocity is a constant $(1.3, 0, 0)$ m/s, and the paddle normal points in the opposite direction of the ball's (x, y)-velocity vector (in order to hit the ball back the way it came).

We ran the default hit plan for 50 trials of topspin balls only to generate an initial swing table of 12 swings. Then—resetting to the initial swing table and hit policy each time—we ran 5 different learning methods on 100 trials of topspin balls and 100 trials of underspin balls, for a total of 10 learning experiments. The policy parameter vector for each learning experiment was $\theta = (x, v_x, v_z, n_y, n_z)$: a subset of the hit plan parameters from section . x was defined as a displacement from the default hit plan's x position, and controlled whether the ball was hit before or after its highest point. Policy parameters were bounded from $(-.1, 1, 0, -.1, -.5)$ to $(.1, 2, .5, .1, .5)$.

Due to the real-time nature of the ping pong task, it was infeasibile to run the learning methods to choose a hit plan θ for every new task parameter vector (i.e., ball parameters) τ. Instead, a lookup table of hit plans was computed between trials for every $\tau_1, \ldots, \tau_n$ the robot has seen in that experiment. Then, when a new incoming ball is detected by the robot (with ball parameters τ_{n+1}), the robot uses the precomputed hit plan from the closest τ_i, or the default hit plan if no nearby τ_i is found.

Rewards were given by the experimenter of 0 or 1 depending on whether the ball was succefully hit back to the other



(a) topspin (b) underspin

Figure 5: Robot ping pong results (cumulative reward vs. learning trials) for topspin and underspin balls.

side of the table. In addition, a coach's pseudo-reward of 0.5 was added if the coach liked the way the robot hit the ball, for a total reward in each trial between 0 and 1.5.

The cumulative reward (without the coach's pseudo-rewards) is shown in figure 5 for each learning method. Using EBDA to include human advice increased reward for both GP-UCB and CrKR for both topspin and underspin. GP-UCB and CrKR are comparable on topspin, while GP-UCB is the clear winner for the harder underspin experiment, which requires a great deal more exploration since the default policy is designed to return topspin balls only.

The results support the idea that global exploration of a reward function (with GP-UCB) is better than local exploration of policy space (with CrKR) when the default policy is far from the optimum. However, the best method may be to combine the two: starting globally and then exploring locally for improvement once a good policy has been found.

Videos of the robot playing ping pong are available at https://sites.google.com/site/aaairobotpingpong/.

Conclusion

We have presented EBDA, a general method for incorporating policy gradient advice into a robot's exploration strategy. By incorporating EBDA into two recent exploration methods, CrKR and GP-UCB, we were able to boost the robot's cumulative reward both in simulation and on a ping pong robot. On the robot, we demonstrated an approach to learning both low-level swing controllers (feed-forward torques) and high-level hit parameters. Our robot was able to learn policies to return both topspin and underspin balls—to our knowledge, this is the first time this has been accomplished, as the underspin "push" is a difficult skill to master, even for human ping pong players.

Acknowledgements

This work was supported in part by the NSF under Grant No. 1117325. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We also gratefully acknowledge support from ONR MURI grant N00014-09-1-1051, from AFOSR grant FA2386-10-1-4135 and from the Singapore Ministry of Education under a grant to the Singapore-MIT International Design Center.

References

Andersson, R. L. 1988. *A robot ping-pong player: experiment in real-time intelligent control*. Cambridge, MA, USA: MIT Press.

Atkeson, C. G., and Schaal, S. 1997. Robot learning from demonstration. In *ICML*, volume 97, 12–20.

Brochu, E.; Cora, V. M.; and De Freitas, N. 2010. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

Contal, E., and Vayatis, N. 2013. Gaussian process optimization with mutual information. *arXiv preprint arXiv:1311.4825*.

Da Silva, B.; Konidaris, G.; and Barto, A. 2012. Learning parameterized skills. *arXiv preprint arXiv:1206.6398*.

Engel, Y.; Mannor, S.; and Meir, R. 2005. Reinforcement learning with gaussian processes. In *In Proc. of the 22nd International Conference on Machine Learning*, 201–208. ACM Press.

Griffith, S.; Subramanian, K.; Scholz, J.; Isbell, C.; and Thomaz, A. L. 2013. Policy shaping: Integrating human feedback with reinforcement learning. In *Advances in Neural Information Processing Systems*, 2625–2633.

Jain, A.; Joachims, T.; and Saxena, A. 2013. Learning trajectory preferences for manipulators via iterative improvement. In *International Conference on Machine Learning (ICML) Workshop on Robot Learning*.

Judah, K.; Roy, S.; Fern, A.; and Dietterich, T. G. 2010. Reinforcement learning via practice and critique advice. In *AAAI*.

Knox, W. B., and Stone, P. 2012. Reinforcement learning from simultaneous human and mdp reward. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 475–482. International Foundation for Autonomous Agents and Multiagent Systems.

Kober, J.; Wilhelm, A.; Oztop, E.; and Peters, J. 2012. Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots* 33(4):361–379.

Kormushev, P.; Calinon, S.; and Caldwell, D. G. 2010. Robot motor skill coordination with EM-based reinforcement learning. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3232–3237. IEEE.

Krause, A., and Ong, C. S. 2011. Contextual gaussian process bandit optimization. In *Advances in Neural Information Processing Systems*, 2447–2455.

Miyazaki, F.; Matsushima, M.; and Takeuchi, M. 2006. Learning to dynamically manipulate: A table tennis robot controls a ball and rallies with a human being. In *Advances in Robot Control*. 317–341.

Mockus, J. 1989. *Bayesian approach to global optimization: theory and applications*. Kluwer Academic Dordrecht, The Netherlands.

Mülling, K.; Kober, J.; Kroemer, O.; and Peters, J. 2013. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research* 32(3):263–279.

Peters, J.; Mülling, K.; Kober, J.; Nguyen-Tuong, D.; and Krömer, O. 2011. Towards motor skill learning for robotics. In *Robotics Research*. Springer. 469–482.

Pilarski, P. M., and Sutton, R. S. 2012. Between instruction and reward: Human-prompted switching. In *2012 AAAI Fall Symposium Series*.

Robbins, H. 1952. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society* 58(5):527–535.

Smart, W. D., and Kaelbling, L. 2002. Effective reinforcement learning for mobile robots. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 4, 3404–3410. IEEE.

Srinivas, N.; Krause, A.; Kakade, S. M.; and Seeger, M. 2009. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*.

Sun, Y.; Xiong, R.; Zhu, Q.; Wu, J.; and Chu, J. 2011. Balance motion generation for a humanoid robot playing table tennis. In *2011 11th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 19–25. IEEE.

Tellex, S.; Kollar, T.; Dickerson, S.; Walter, M. R.; Banerjee, A. G.; Teller, S. J.; and Roy, N. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*.

Thomaz, A. L., and Breazeal, C. 2008. Teachable robots: Understanding human teaching behavior to build more effective robot learners. *Artificial Intelligence* 172(6):716–737.