Peg Insertion with Policy Search

Dynamic Programming Final Project
Caris Moses

January 16, 2018

1 Introduction

A common manipulation task in robotics is peg insertion. In my formulation of the problem force control is used maneuver a peg into a slot slightly wider than the peg. Figure 1 gives a visual of my domain. I will give results for various model-free policy search methods which aim to learn the parameters of a force controller, or policy. Model-free policy search methods provide a generalizable method for learning parameterized policies.

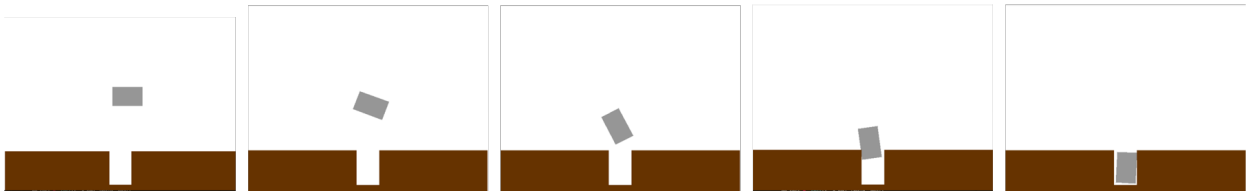

Figure 1: A successful learned peg insertion task.

2 Policy Formulations

A policy, π, is a mapping from a state, $\mathbf{s}$, to an action, $\mathbf{a}$. A policy can be either deterministic or stochastic. Using a stochastic policy can be beneficial as it forces a system to both exploit its knowledge as well as perform some exploration of the action or parameter space in hopes of finding better parameters.

2.1 Linear Stochastic Policy

A parameterized linear stochastic policy takes the following form.

$$\pi(\mathbf{a}|\mathbf{s};\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}(\mathbf{s}-\mathbf{g}), \Sigma) \tag{1}$$

The mean of the policy's distribution is linear in the state of the peg, $\mathbf{s} \in \mathbf{R}^n$. The goal state is $\mathbf{g} \in \mathbf{R}^n$. Actions are given by $\mathbf{a} \in \mathbf{R}^m$ making $\boldsymbol{\theta}$ an $m \times n$ matrix. The noise in the policy is $\epsilon \sim \mathcal{N}(\mathbf{0}, \Sigma)$ where Σ is a $m \times m$ covariance matrix.

The following cost function is used with the linear stochastic policy. It is simply the distance from the goal state at time t.

$$c_t = ||\mathbf{s}_t - \mathbf{g}||^2 \tag{2}$$

The cost of a trajectory is therefore

$$C(\tau) = \sum_{t=0}^{T} c_t \tag{3}$$

2.2 Dynamic Movement Primitives

Dynamic Movement Primitives [3], or DMPs, give a point attractor policy with an added forcing function. I will only consider DMPs for a 1-dimensional action, a_s, which acts on the pose, s, and velocity, v_s, of a 1-dimensional system. DMPs are characterized by a parameter $\boldsymbol{\theta}_s$.

2.2.1 Deterministic DMP

To begin I will give the deterministic formulation of a DMP.

$$\pi(a_s|s, v_s; \boldsymbol{\theta}_s) = -K(s - g_s) - Dv_s + \mathbf{b}^T\boldsymbol{\theta}_s \tag{4}$$

The first 2 terms in the policy characterize a PD controller which is driving the pose of the system, s, to g_s. K is the spring coefficient and D is the damping coefficient.

DMPs also utilize a canonical system. The state of the canonical system, z, is defined by the following simple dynamics.

$$\dot{z}_t = -\alpha z_t \tag{5}$$

where α is a time constant and $z_0 = 1$. This system decays exponentially to 0 as t goes to infinity. This canonical system is used in the forcing term, $\mathbf{b}^T\boldsymbol{\theta}_s$ of the policy. $\mathbf{b}$ is a basis function of p radial basis function kernels. Therefore $\mathbf{b}$ and $\boldsymbol{\theta}$ are $p \times 1$ vectors. The jth term of $\mathbf{b}$ is as follows

$$[\mathbf{b}]_j = \frac{\psi_j}{\sum_{i=1}^{p} \psi_i} z_t |s_0 - g_s| \tag{6}$$

$$\psi_j = \exp(-h_j(z_t - c_j)^2) \tag{7}$$

where $c_j \in [0, 1]$. This means that the j^{th} kernel is "activated" when z_t is near c_j. These kernels are then multiplied by the parameter, $\boldsymbol{\theta}_s$, which acts as a weight on the kernels. For example if we want a large force to act on the system early on in the trajectory, then we would have a kernel with c_j close to 1 and the jth term of $\boldsymbol{\theta}$ with the desired force we want to enforce on our system. $|s_0 - g_s|$ is a spatial scaling term with s_0 being the initial pose of the system.

It is important to notice that without the forcing term ($\mathbf{b}^T\boldsymbol{\theta}_s$) the DMP policy is simply a PD controller. In addition, the affects of the forcing term decay to zero as t goes to infinity. This is due to the z_t term in the basis function, $\mathbf{b}$. This means that as t goes to infinity and z_t goes to 0 the policy reverts back to the PD controller which will drive the system to the goal state.

2.2.2 Parameter Space Exploration

The DMP is converted into a stochastic policy by adding noise to the parameter vector $\boldsymbol{\theta}_s$. The stochastic policy takes the following form.

$$\pi(a_s|s, v_s; \boldsymbol{\theta}_s) = -K(s - g_s) - Dv_s + \mathbf{b}^T(\boldsymbol{\theta}_s + \boldsymbol{\epsilon}) \tag{8}$$

$$\epsilon \sim \mathcal{N}(\mathbf{0}, \Sigma) \tag{9}$$

This stochastic policy is equivalent to the following representation.

$$\pi(F_s|s, v_s; \boldsymbol{\theta}_s) = \mathcal{N}(-K(s - g_s) - Dv_s + \mathbf{b}^T\boldsymbol{\theta}_s, \mathbf{b}^T\Sigma\mathbf{b}) \tag{10}$$

This method enforces parameter space exploration by adding noise to the parameter vector at each step in the trajectory. The cost function used for this DMP policy takes the following form.

$$c_t = q_t + (\boldsymbol{\theta}_s + \epsilon_t)^T\mathbf{R}(\boldsymbol{\theta}_s + \epsilon_t) \tag{11}$$

$$q_t = Q|s_t - g_s| \tag{12}$$

The cost has a state dependent term, q_t as well as a regularization term on the control input from the forcing function. Q is a scalar constant and R is a $p \times p$ matrix. This form of the policy and cost function is required for a method of policy search (Policy Improvement with Path Integrals) which will be discussed in Section 3.2.

2.2.3 Action Space Exploration

The DMP can also be converted into a stochastic policy which enforces action space exploration. The stochastic policy takes the following form.

$$\pi(a_s|s, v_s; \boldsymbol{\theta}_s) = -K(s - g_s) - Dv_s + \mathbf{b}^T\boldsymbol{\theta}_s + \epsilon \tag{13}$$

$$\epsilon \sim \mathcal{N}(\mathbf{0}, \Sigma) \tag{14}$$

This stochastic policy is equivalent to the following representation.

$$\pi(F_s|s, v_s; \boldsymbol{\theta}_s) = \mathcal{N}(-K(s - g_s) - Dv_s + \mathbf{b}^T\boldsymbol{\theta}_s, \Sigma) \tag{15}$$

This method enforces action space exploration by adding noise to the action selected at each step in the trajectory. The cost function used for this DMP policy takes the following form.

$$c_t = q_t + \boldsymbol{\theta}_s^T\mathbf{R}\boldsymbol{\theta}_s \tag{16}$$

$$q_t = Q|s_t - g_s| \tag{17}$$

This form of the policy and cost function is required for a method of policy search (REINFORCE) which will be discussed in Section 3.1.2.

3 Model-free Policy Search Methods

Policy search methods aim to find policy parameters, $\boldsymbol{\theta}$, which best performs the task and minimize the cost. Model-free methods are able to do so without a model of the system dynamics. First, I will give two methods which minimize the cost via gradient descent; Finite Differences and REINFORCE. Then, I will give a third method, Policy Improvement with Path Integrals.

3.1 Gradient Descent

A trajectory, τ, is the sequence of states and actions that a system takes for a given time horizon of length T.

$$\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, ..., \mathbf{s}_T) \tag{18}$$

The total accumulated cost over a trajectory is

$$C(\tau) = \sum_{t=0}^{T} c_t \tag{19}$$

Gradient descent policy search methods aim to find the policy parameters which minimize the value function, $J(\boldsymbol{\theta})$, or the expectation of the cost function.

$$J_{\boldsymbol{\theta}} = \mathbf{E}_\tau[C(\tau)|\boldsymbol{\theta}] \tag{20}$$

For gradient descent methods the parameter update takes the following form.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}J_{\boldsymbol{\theta}} \tag{21}$$

where η is the learning rate and $\nabla_{\boldsymbol{\theta}}J_{\boldsymbol{\theta}}$ is the gradient with respect to parameter, $\boldsymbol{\theta}$.

3.1.1 Finite Differences

A basic estimate of this gradient is a finite difference. In summary this works by perturbing each dimension of your parameter by some δ and $-\delta$, and estimating the gradient as the change in your value function. For a 1-dimensional parameter, θ, the gradient estimate would look like this

$$\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}} \approx \frac{\hat{J}(\theta+\delta) - \hat{J}(\theta-\delta)}{2\delta} \tag{22}$$

where we calculate $\hat{J}(\theta)$ by averaging the cost, $C(\tau)$, over several trajectories using the parameter θ [4].

Now we are given a parameter, $\boldsymbol{\theta}$ which contains l elements. We generate a matrix $\Delta\Theta$ which is $N \times l$ where N is the number of trajectories we wish to run to estimate the gradient. The following $\Delta\Theta$ would be for $N = l$. In practice, for stochastic systems N can and should be larger then l to ensure that each element of the parameter uses more than 2 trajectories to estimate the gradient with respect to the that element of the parameter.

$$\Delta\Theta = \begin{bmatrix} 2\delta & 0 & \dots & 0 \\ 0 & 2\delta & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & 2\delta \end{bmatrix}^T \tag{23}$$

For this method we only perturb one element of the parameter vector at a time. Other methods exist which perturb more than one element of the parameter at a time.

Now we construct $\Delta\hat{J}$ which is the estimate of the change in the value function between $+\delta$ and $-\delta$. $\Delta\hat{J}$ is an $l \times 1$ vector where $[\Delta\hat{J}]_j$ is $\hat{J}(\boldsymbol{\theta} + \boldsymbol{\delta}_j) - \hat{J}(\boldsymbol{\theta} - \boldsymbol{\delta}_j)$ and $\boldsymbol{\delta}_j$ is a vector or 0s with the jth element set to δ.

Finally, the gradient estimate takes the following form.

$$\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}} \approx (\Delta\Theta^T \Delta\Theta)^{-1} \Delta\Theta^T \Delta\hat{J} \tag{24}$$

To update the parameter vector, $\boldsymbol{\theta}$, it matrix must be converted into an $l \times 1$ vector.

3.1.2 REINFORCE

REINFORCE [8] is another gradient estimation method of policy search. It uses the log ratio trick to get the expectation of the gradient of the cost function into something that can be estimated via sampling.

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}} &= \mathbf{E}_\tau[C(\tau)] \\ &= \int_\tau C(\tau) \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\tau) d\tau \\ &= \int_\tau C(\tau) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) p_{\boldsymbol{\theta}}(\tau) d\tau \\ &= \mathbf{E}_\tau[C(\tau) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau)] \end{aligned} \tag{25}$$

where $p_{\boldsymbol{\theta}}(\tau)$ is the probability of a trajectory, τ, under parameter $\boldsymbol{\theta}$. The derivation above uses the log ratio trick, meaning we use the fact that

$$\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) = \frac{\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\tau)}{p_{\boldsymbol{\theta}}(\tau)} \tag{26}$$

Then we must calculate the probability of a trajectory under a stochastic policy.

$$p_\theta(\tau) = p(\tau|\mathbf{s}_0; \boldsymbol{\theta}) = p(\mathbf{s}_0) \prod_{t=1}^{T} p(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1}) \pi(\mathbf{a}_{t-1}|\mathbf{s}_{t-1}; \boldsymbol{\theta}) \tag{27}$$

where $p(\mathbf{s}_0)$ is the probability of starting the trajectory in state $\mathbf{s}_0$. When we calculate $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau)$ from the distribution above (in Equation 27). Since only the policy term depends on $\boldsymbol{\theta}$ the remaining probability terms drop out. This is what allows REINFORCE to be model-free. The transition probability does not need to be known the estimate the gradient on the cost function.

$$\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) = \nabla_{\boldsymbol{\theta}} \log \pi(\mathbf{a}|\mathbf{s}; \boldsymbol{\theta}) \tag{28}$$

As you can imagine the trajectories used to estimate the gradient of the cost function are very high variance. The noise from each step within the trajectory adds up. Often a baseline, b, is used to reduce the variance of the gradient estimate. The gradient then takes the following form

$$\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}} = \mathbf{E}_{\tau}[(C(\tau) - b)\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau)] \tag{29}$$

where $b \in \mathbf{R}$ is the baseline. This does not introduce bias into the gradient estimate calculation as

$$\mathbf{E}_{\tau}[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) b] = b \int_{\tau} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) p_{\boldsymbol{\theta}}(\tau) d\tau = \int_{\tau} \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\tau) = 0 \tag{30}$$

Here we use the log ratio trick (Equation 26) again, as well as the fact that $\int_{\tau} p_{\boldsymbol{\theta}}(\tau) d\tau = 1$ [5]. This proves that for infinite data the baseline will vanish. This baseline estimates the optimal baseline which is the minimizer of the variance of the gradient estimates with respect to a baseline. See [9] for more details. The baseline used in my work is as follows [9].

$$b = \frac{\mathbf{E}_{\tau}[C(\tau)||\sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s};\boldsymbol{\theta})||^2]}{\mathbf{E}_{\tau}[||\sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s};\boldsymbol{\theta})||^2]} \tag{31}$$

Using K trajectories each of length T to estimate the gradient, the update step takes the following form.

$$\begin{aligned} \boldsymbol{\theta} &= \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}} \\ &= \boldsymbol{\theta} - \eta \mathbf{E}_{\tau}[(C(\tau) - b)\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau)] \\ &= \boldsymbol{\theta} - \eta \frac{1}{K} \sum_{k=0}^{K} (C(\tau_k) - b) \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi(\mathbf{a_t}|\mathbf{s_t};\boldsymbol{\theta}) \end{aligned} \tag{32}$$

3.2 Policy Improvement Path Integrals

Policy Improvement Path Integrals [2], [7], or PI^2, utilizes the path integral approach to optimal control. Path integrals give an optimal control law for nonlinear systems. This optimal control law is then used to derive the parameter update for the PI^2 algorithm. PI^2 is a method of policy search referred to as probability weighted averaging. It is different from gradient approximation methods as it is not trying to estimate a gradient, and therefore should not face the same challenged as gradient descent methods (local minima, noisy estimates of the gradient, etc...).

3.2.1 Nonlinear System

The discrete-time nonlinear continuous system for a state $\mathbf{s}$ is defined as follows.

$$\begin{aligned} \dot{\mathbf{s}}_t &= f(\mathbf{s}_t) + G(\mathbf{s}_t)(\mathbf{a}_t + \epsilon) \\ &= f_t + G_t(\mathbf{a}_t + \epsilon_t) \end{aligned} \tag{33}$$

where f_t represents the passive dynamics, G_t is the potentially state dependent control matrix, and $\epsilon \sim \mathcal{N}(\mathbf{0}, \Sigma)$ is noise. The equivalent discrete time system is given by the following

$$\mathbf{s}_{t+dt} = \mathbf{s}_t + f_t dt + G_t(\mathbf{a}_t dt + \epsilon_t dt) \tag{34}$$

The discrete time transition probability is given by

$$p(\mathbf{s}_{t+dt}|\mathbf{s}_t, \mathbf{a}_t) = \mathcal{N}(\mathbf{s}_t + f_t dt - G_t \mathbf{a}_t dt, \Sigma_a dt) \tag{35}$$

where $\Sigma_a = G_t \Sigma G_t^T$.

3.2.2 Hamilton Jacobi Bellman Equation

The discrete time Bellman equation for this system (with the cost function given in Equations 16 and 17) is given by

$$J(\mathbf{s}, t) = r_t(\mathbf{s}_t dt) + \max_a\{-\mathbf{a}_t^T R_t \mathbf{a}_t dt + \mathbf{E}_{s_{t+dt}}[J(\mathbf{s}_{t+dt}, t + dt]\} \tag{36}$$

The discrete time Bellman equation can be approximated as a 2^{nd} order Taylor series. Since the cost function is quadratic, the expectation can be computed analytically. Then, by taking the limit as $dt \to 0$ the Hamilton Jacobi Bellman Equation can be derived

$$-\dot{J}(\mathbf{s},t) = r_t(\mathbf{s}_t) + \max_a\{-\mathbf{a}_t^T R_t \mathbf{a}_t + (f_t + G_t\mathbf{a}_t)^T \mathbf{j}_s + \frac{1}{2}\text{tr}(\Sigma_a \mathbf{J}_{ss})\} \tag{37}$$

where $\mathbf{j}_a = \frac{\partial}{\partial s} J(\mathbf{s}, t+dt)$ and $\mathbf{J}_{ss} = \frac{\partial}{\partial^2 s} J(\mathbf{s}, t+dt)$.

3.2.3 Optimal Control and Path Integral

To determine the optimal control which minimizes the cost of the Hamilton-Jacobi Bellman equation, take the derivative of 37 with respect to $\mathbf{a}$ and set it equal to 0. This gives the following optimal control law

$$\mathbf{a}_t = R^{-1} G_t^T \mathbf{j}_s \tag{38}$$

By plugging this back in to Equation 37 we get the Hamilton-Jacobi Bellman equation under the optimal control law

$$-\dot{J}(\mathbf{s},t) = r_t(\mathbf{s}_t) + \mathbf{j}_s^T G_t R^{-1} G_t^T \mathbf{j}_s + \mathbf{j}_s^T f_t + \frac{1}{2}\text{tr}(\mathbf{J}_{ss}\Sigma_a) \tag{39}$$

Equation 39 is a nonlinear 2^{nd} order partial differential equation. By using the following exponential transformation of the value function we can get a set of linear partial differential equations.

$$\Psi(\mathbf{s},t) = \exp(\frac{J(\mathbf{s},t)}{\lambda}) \tag{40}$$

where λ represents the "temperature" of the transformation. With this transformation, the assumption that $R = \lambda\Sigma_a^{-1}$, and the use of the Feynman-Kac theorem for solving partial differential equations, a solution for Ψ is obtained.

$$\Psi(\mathbf{s},t) = \mathbf{E}_\tau[-\frac{1}{\lambda}\int_0^T q_t dt] \tag{41}$$

We define the path integral as the following under the assumption that the control matrix, G_t, is state independent.

$$S(\tau) = \sum_{t=0}^{T} q_t dt + \frac{1}{2}\sum_{t=0}^{T-1} ||\frac{\mathbf{s}_{t+1} - \mathbf{s}_t}{dt}||^2_{\mathbf{H}_t^{-1}} dt \tag{42}$$

$$\mathbf{H}_t = G_t R^{-1} G_t^T \tag{43}$$

The solution to Ψ becomes

$$\Psi(\mathbf{s},t) = \lim_{dt\to 0} \int \exp(-\frac{1}{\lambda}S(\tau) + \frac{\lambda}{2}\sum_{t=0}^{T}\log|\mathbf{H}_t|) \tag{44}$$

The optimal control law after the exponential transformation takes the following form and can be further simplified using Equation 44.

$$\mathbf{a}_t = \lambda R^{-1} G_t^T \frac{\nabla_{\mathbf{s}}\Psi(\mathbf{s},t)}{\Psi(\mathbf{s},t)_t} \quad = \int P(\tau)\mathbf{a}_L(\tau) s\tau \tag{45}$$

where

$$P(\tau) = \frac{\exp(-\frac{1}{\lambda}S(\tau))}{\int \exp(-\frac{1}{\lambda}S(\tau))d\tau} \tag{46}$$

The form of $\mathbf{a}_L$ depends on the system being controlled. The definition of $\mathbf{a}_L$ will be given below in Equation 47.

3.2.4 Policy Improvement Path Integrals

Now that we have the form of the optimal control using path integrals, we will show how to develop a reinforcement learning algorithm based on this method. We will use the policy given in Equation 8, which is a DMP with parameter space exploration noise. Notice that this takes the same form as the control system given in Equation 33 where $f_t = -K(s - g_s) - Dv_s$, $G_t = \mathbf{b}^T$, and $\mathbf{a}_t = \boldsymbol{\theta}$. $\mathbf{b}^T \in \mathbf{R}^p$ where p is the number of basis functions.

For this case (a 1-dimension state being directly controlled, and the control matrix, $\mathbf{b}$, is state independent) the optimal control law from Equation 45 takes the following form.

$$\mathbf{a}_L = \frac{R^{-1}\mathbf{b}_t}{\mathbf{b}_t^T R^{-1}\mathbf{b}_t}(\mathbf{b}_t^T \epsilon_t) \tag{47}$$

For this system the path integral for a trajectory starting at time t from Equation 42 takes the following form

$$S(\tau_t) = \sum_{t'=t}^{T} q_{t'} + \frac{1}{2}(\boldsymbol{\theta} + \epsilon_{t'})^T M_{t'}^T R M_{t'}(\boldsymbol{\theta} + \epsilon_{t'}) \tag{48}$$

where

$$M_t = \frac{R^{-1}\mathbf{b}_t\mathbf{b}_t^T}{\mathbf{b}_t^T R^{-1}\mathbf{b}_t} \tag{49}$$

Next will introduce $P(\tau_t)$ which is the probability of a trajectory τ starting at time t.

$$P(\tau_t) = \frac{\exp(-\frac{1}{\lambda}S(\tau_t))}{\sum_\tau \exp(-\frac{1}{\lambda}S(\tau_t))} \tag{50}$$

This is a softmax which ensures that the sum over $P(\tau_t)$ over all sampled trajectories at time t sums to 1. Paths with higher path costs with have higher $P(\tau_t)$ values.

Using the optimal control law given by Equations 45 and 47, the update rule for a parameter $\boldsymbol{\theta}$ at time t will take the following form

$$\delta\boldsymbol{\theta}_t = \sum_\tau P(\tau_t)\mathbf{M}_t\epsilon_t \tag{51}$$

However this gives an update for all t. To calculate an update for each dimension, j, of the parameter, we average over all $\delta\boldsymbol{\theta}_t$. This averaging weights early times more heavily as they have a higher influence over the overall trajectory.

$$[\delta\boldsymbol{\theta}]_j = \frac{\sum_{t=0}^{T}(T-t)\psi_{j,t}[\delta\boldsymbol{\theta}_t]_j}{(T-t)\psi_{j,t}} \tag{52}$$

Finally, the update to $\boldsymbol{\theta}$ occurs as follows

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \delta\boldsymbol{\theta} \tag{53}$$

By using the following equality you can eliminate the parameter λ from the algorithm all together.

$$\exp(-\frac{1}{\lambda}S(\tau_t)) = \exp(-h\frac{S(\tau_t) - \min S(\tau_t)}{\max S(\tau_t) - \min S(\tau)_t}) \tag{54}$$

See [7] for more details.

4 Problem Description

The state consists of the pose and velocity of the peg, $\mathbf{s} = [x, y, \theta, v_x, v_y, v_\theta]^T$. Note that the bold $\boldsymbol{\theta}$ refers to the parameter and the unbolded θ refers to the θ-position of the peg in $\mathbf{s}$. The agent can act on the peg by applying forces in the x and y-directions as well as a torque in the θ-direction, making the output of the policy $\mathbf{a} = [F_x, F_y, F_\theta]^T$. The goal state of the peg is specified as a vector of desired values for each of the peg states, $\mathbf{g} = [g_x, g_y, g_\theta, 0, 0, 0]^T$.

To simulate realistic agent movements and the interaction between the peg and the hole I used pybox2d [1] as a physics engine. I used pygame [6] to render the objects in a simulation environment. Both pygame and pybox2d are typically used for game development in python, but I found them well suited for this simplified robotics simulation.

4.1 Policies Implemented

In the following section I will compare the performance of Finite Differences, REINFORCE, and PI^2. First I would like to note which policies were used in these implementations. PI^2 required the policy to be parameterized as a DMP with exploration in the parameter space given in Equation 8. REINFORCE required the DMP to perform action space exploration, as given in Equation 13. This is due to the fact that adding time dependent noise to the parameter vector at each time step (Eq 8) introduced too much variance into the trajectories to be successful in a REINFORCE policy search (even when using an optimal baseline). In addition I was only able to compare the performance of Finite Differences and REINFORCE using a linear stochastic policy due to the limitations of PI^2.

5 Results & Analysis

5.1 Finite Differences, REINFORCE, and PI^2 with DMPs

5.1.1 Parameter Update Visualizations

In order to visualize the parameter space and the updates to the parameters made in that space, I simplified the problem to only use a parameter, $\boldsymbol{\theta} \in \mathbf{R}^2$. $\boldsymbol{\theta} = [\boldsymbol{\theta}_y, \boldsymbol{\theta}_\theta]$ where $\boldsymbol{\theta}_y, \boldsymbol{\theta}_\theta \in \boldsymbol{R}$. The peg starts at $[x, y, z] = [0, 4, 0]$ and has to end in the hole at $[g_x, g_y, g_\theta] = [0, -9, \frac{-\pi}{2}]$. The policies used for each dimension of the state are given below

$$\pi(F_x|x, v_x) = -K(x - g_x) - Dv_x \tag{55}$$

$$\pi(F_y|y, v_y; \boldsymbol{\theta}_y) = \mathcal{N}(-K(y - g_y) - Dv_y + \mathbf{b}^T\boldsymbol{\theta}_y, \mathbf{b}^T\boldsymbol{\Sigma}\mathbf{b}) \tag{56}$$

$$\pi(F_\theta|\theta, v_\theta; \boldsymbol{\theta}_\theta) = \mathcal{N}(-K(\theta - g_\theta) - Dv_\theta + \mathbf{b}^T\boldsymbol{\theta}_\theta, \mathbf{b}^T\boldsymbol{\Sigma}\mathbf{b}) \tag{57}$$

Figures 2, 3, and 4 show the update steps each algorithm would take if it used 30 trajectories to estimate the update step. All parameters used in these runs are give in Table 1. The color plot in the background of these figures gives a visual of the value function. It is calculated by running a trajectory for each parameter in the figure with no exploration noise, and making an intensity plot of the costs returned by those trajectories (using Equations 16 and 17).

As for the arrows in Figures 2 (FD) and 3 (REINFORCE), they represent the estimate of the gradient of the value function at the parameters corresponding to the root of the arrow. For Figure 4 (PI^2) the arrows represent the $\delta\boldsymbol{\theta}$ update which the PI^2 algorithm would make at the parameter setting corresponding to the root of the arrow. Lastly, the white dot represents the minima of the cost function which corresponds to a successful peg insertion starting from the initial position with no exploration noise.

Table 1: **Parameters Used in Stochastic DMP Trajectories**

Steps in a trajectory, T	300
Trajectories per parameter update vector calculation	30 (Fig 2 - 4), 5 (Fig 5[])
Spring Constant, K	1
Damping Constant, D	0.5
RBF params, c_0, h_0	0.95, 2
Learning Rate, η	1e-8
Exploration noise, Σ	0. (Fig 2 - 4), .001 (Fig 5[])
Final distance from goal to consider task learned	0.5
State dependent cost coefficient, Q	100
Control cost matrix, R	1
FD-specific, δ	1
PI^2-specific h	10
Granularity of value function intensity plot	0.2
Granularity of update vector calculation	2

Just by observation, Figures 2 - 4 all show that in the steeper areas of the value function (the green portions), all methods are able to give updates in the correct direction towards the minima. The areas in which all methods seem to have trouble is when the cost function flattens out (the purple portions). It appears that PI^2 has the noisiest updates in terms of the $\delta\boldsymbol{\theta}$ vectors not consistently pointing towards the minima of the cost function.

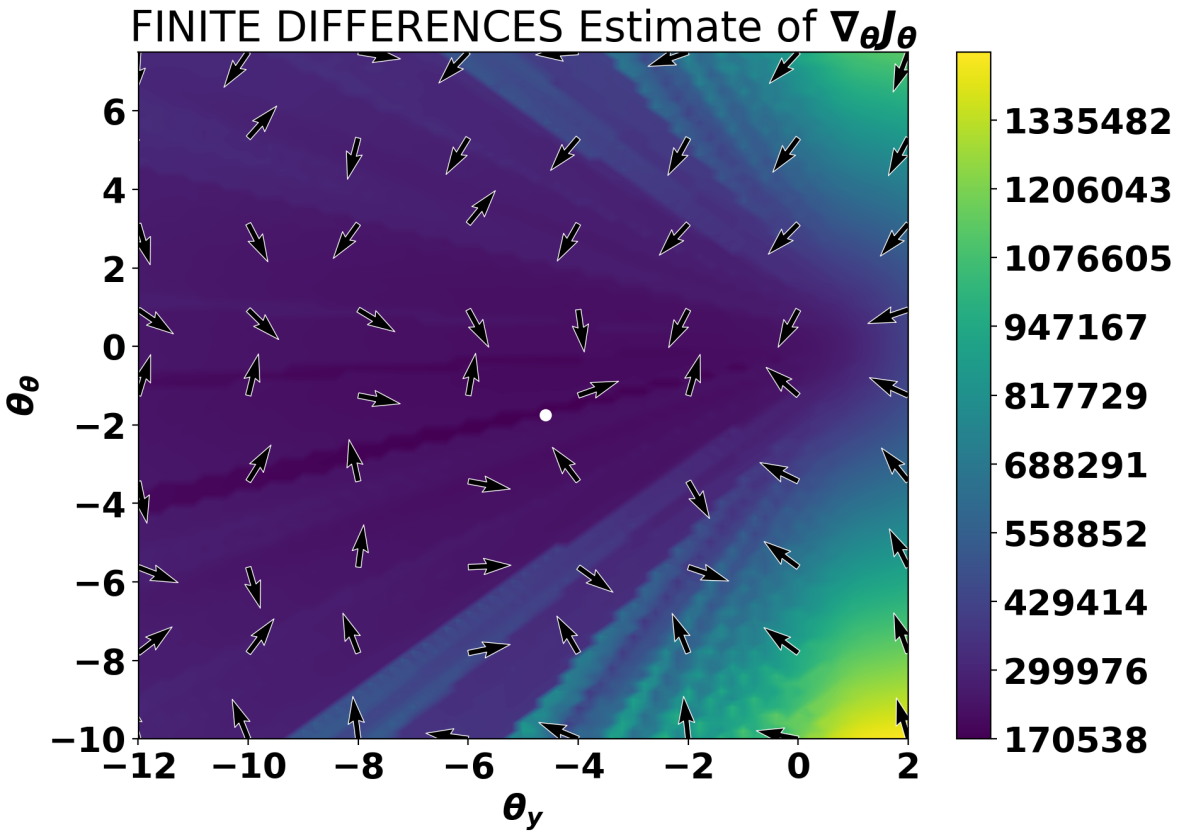

Figure 2: The intensity plot in the background is the empirical value function generated by running trajectories with no exploration noise. The white dot represents the minima of this value function. The vectors are estimates of the gradient using Finite Differences. All parameters used to generate this plot are given in Table 1.

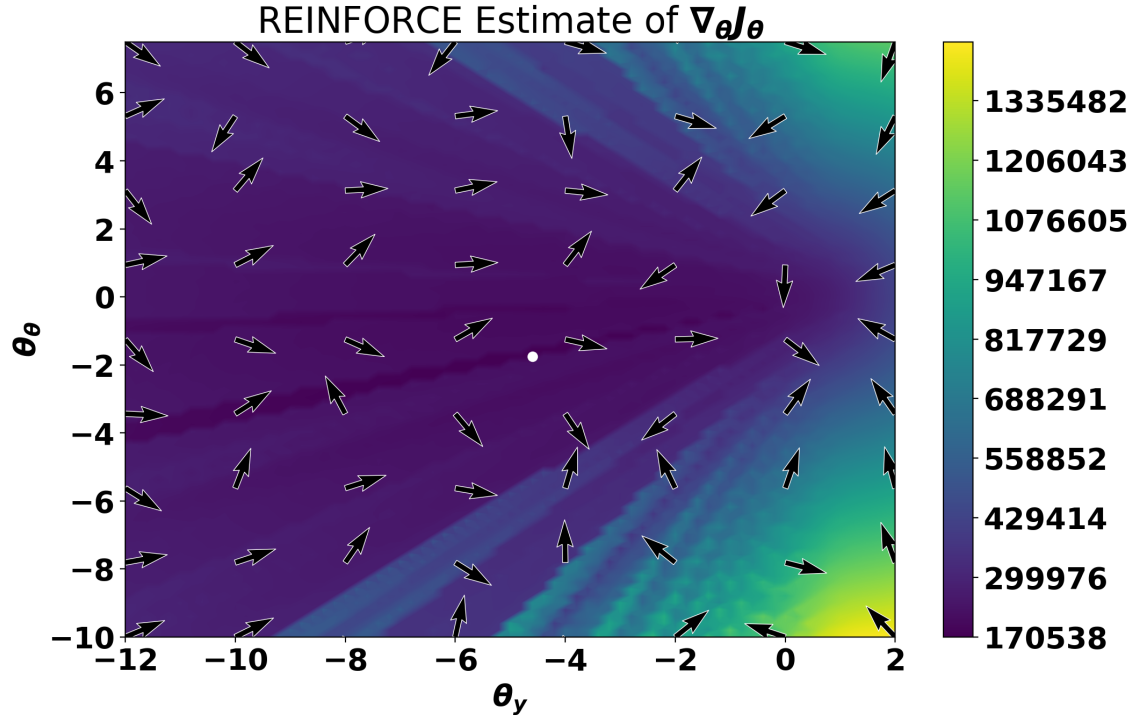

Figure 3: The intensity plot in the background is the empirical value function generated by running trajectories with no exploration noise. The white dot represents the minima of this value function.The vectors are estimates of the gradient using REINFORCE. All parameters used to generate this plot are given in Table 1.

5.1.2 Parameter Learning

Figure 5 shows how each algorithm performed with the same initial parameter. As you can see REINFORCE outperformed PI^2 which outperformed Finite Differences. The algorithms all end when the task is learned. You can see that REINFORCE converges on a small cost quickly, as well as learns the task before the other two methods. The parameters used during these runs are also given in Table 1.

The number of trajectories needed to learn the insertion task is 5 $\times$ the number of iterations given in Figure 5[left] as each step averaged over 5 trajectories.

5.2 Finite Differences and REINFORCE with Linear Stochastic Policy

As mentioned earlier, I could only compare FD and REINFORCE with the linear stochastic policy as PI^2 is restricted to using the DMP policy parameterization. The linear stochastic policy in given in Equation 1, and uses the following representations.

$$\mathbf{F} = \begin{bmatrix} F_x \\ F_y \\ F_\theta \end{bmatrix}, \mathbf{s} = \begin{bmatrix} x \\ y \\ \theta \\ v_x \\ v_y \\ v_\theta \end{bmatrix}, \mathbf{g} = \begin{bmatrix} g_x \\ g_y \\ g_\theta \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{58}$$

Figure 5[right] shows the performance of FD and REINFORCE gradient descent with this policy parameterization. The parameters used for these runs are given in Table 2. As you can see REINFORCE is able to find a low cost region quicker, which FD learns the task first (the descent ends earlier for FD). However, since this is a stochastic system, I don't believe that the fact that FD converged first is significant. There are instances when REINFORCE learns the task first, but it consistently converges quicker than FD.

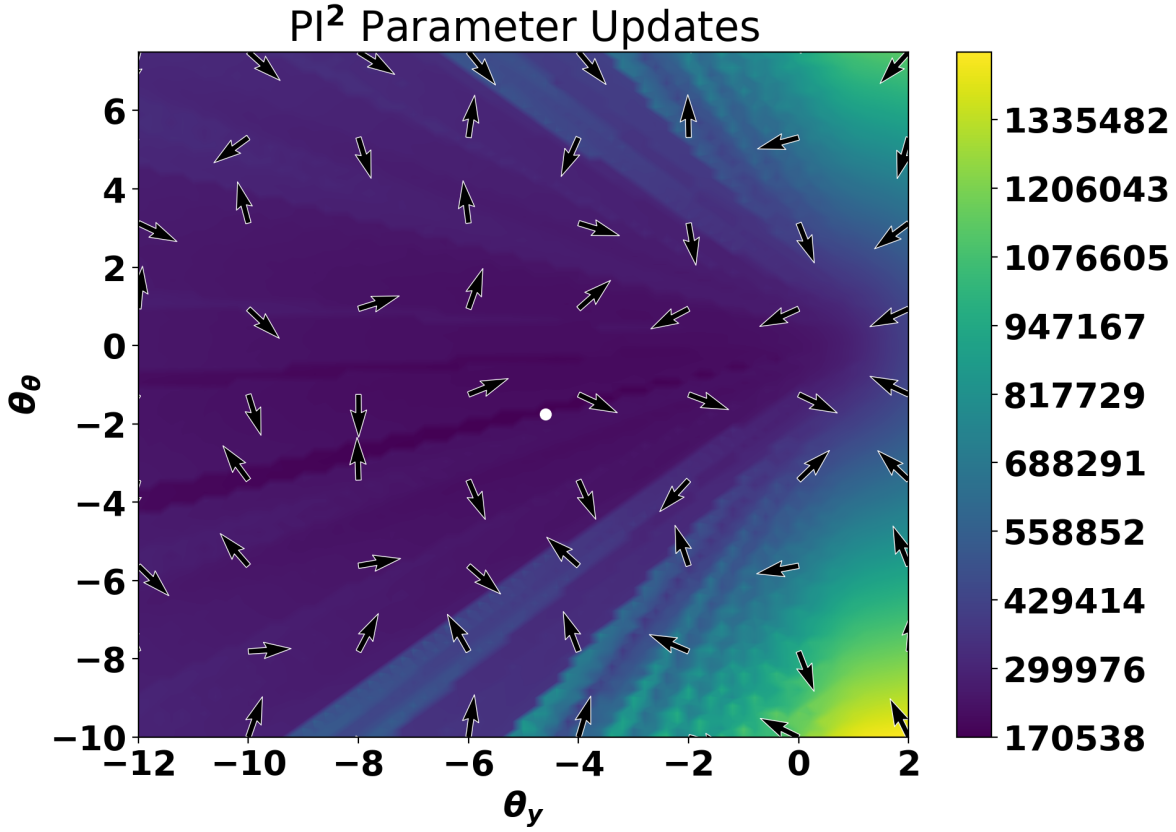

Figure 4: The intensity plot in the background is the empirical value function generated by running trajectories with no exploration noise. The white dot represents the minima of this value function. The vectors are update vectors, $\delta\boldsymbol{\theta}$ generate by the PI^2 algorithm.All parameters used to generate this plot are given in Table 1.

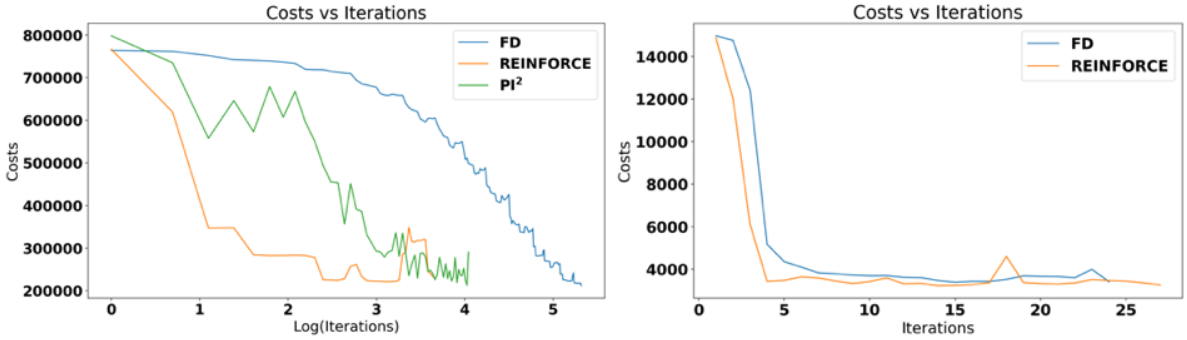

Figure 5: The figure on the shows the performance of the algorithms (FD, REINFORCE, and PI^2) using a stochastic DMP policy. The plot on the right shows the performance of FD and REINFORCE using a linear stochastic policy.

Table 2: **Parameters Used in Linear Stochastic Policy Trajectories**

Steps in a trajectory, T	500
Learning Rate, η	1e-5
Exploration noise, Σ	$10 \times \mathrm{I}_{m \times m}$
Final distance from goal to consider task learned	0.5
FD-specific δ	1
PI^2-specific h	10

6 Analysis & Conclusion

The policies learned by these algorithms are very unstable. By only using $p = 1$ basis function kernel, the learned policy has only "one chance" during a trajectory to apply a force which will result in a successful insertion. In addition, since I selected $c_0 = .95$, this applied force occurs very early on in the trajectory. For highly stochastic systems this is unlikely to be successful as errors in the controls and system dynamics accumulate over time and will must likely deviate from the predicted trajectory.

I planned on testing out these methods with higher dimensional parameter vectors however I found they they were *very* sensitive to the initial parameters. The higher dimensional the parameter is, the more local minima induced on your value function. Therefore I was unable to learn high dimensional parameters to perform a peg insertion.

In addition, DMPs are parameterized in a time-dependent manner. The RBF kernels are activated as a function of time. Therefore if an outside or unpredicted force acts on the system, the learned policy will not be able to correct for this. I believe that the fact that these methods are so sensitive to parameter initializations and the time dependence DMPs induce make them ill-fitted for the peg insertion task. I think that the linear stochastic policy is much better suited for the task as the learned parameters are state dependent and not time dependent. Therefore they can perform reasonable actions (if learned correctly) anywhere in the state space at any time no matter what outside forces interact with the system.

References

[1] Erin Catto. pybpx2d. https://github.com/pybox2d/pybox2d.

[2] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013.

[3] Travis DeWolf. Dynamic movement primitives part 1: The basics. https://studywolf.wordpress.com/2013/11/16/dynamic-movement-primitives-part-1-the-basics/, 2013.

[4] Travis DeWolf. Simultaneous perturbation vs finite differences for linear dynamics estimation and control signal optimization. https://studywolf.wordpress.com/tag/finite-differences/, 2016.

[5] Jan Peters. Policy gradient methods. http://www.scholarpedia.org/article/Policy_gradient_methods, 2010.

[6] Pete Shinners. Pygame. http://pygame.org/, 2011.

[7] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research*, 11(Nov):3137–3181, 2010.

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

[9] Tingting Zhao, Hirotaka Hachiya, Gang Niu, and Masashi Sugiyama. Analysis and improvement of policy gradient estimation. In *Advances in Neural Information Processing Systems*, pages 262–270, 2011.