8. Appendix

8.1. Derivation of LQR-FLM

Given a TVLG dynamics model and quadratic cost approximation, we can approximate our Q and value functions to second order with the following dynamic programming updates, which proceed from the last time step $t = T$ to the first step $t = 1$:

$$Q_{\mathbf{x},t} = c_{\mathbf{x},t} + \mathbf{f}_{\mathbf{x},t}^\top V_{\mathbf{x},t+1}\,,\ \ Q_{\mathbf{xx},t} = c_{\mathbf{xx},t} + \mathbf{f}_{\mathbf{x},t}^\top V_{\mathbf{xx},t+1}\mathbf{f}_{\mathbf{x},t}\,,$$
$$Q_{\mathbf{u},t} = c_{\mathbf{u},t} + \mathbf{f}_{\mathbf{u},t}^\top V_{\mathbf{x},t+1}\,,\ \ Q_{\mathbf{uu},t} = c_{\mathbf{uu},t} + \mathbf{f}_{\mathbf{u},t}^\top V_{\mathbf{xx},t+1}\mathbf{f}_{\mathbf{u},t}\,,$$
$$Q_{\mathbf{xu},t} = c_{\mathbf{xu},t} + \mathbf{f}_{\mathbf{x},t}^\top V_{\mathbf{xx},t+1}\mathbf{f}_{\mathbf{u},t}\,,$$
$$V_{\mathbf{x},t} = Q_{\mathbf{x},t} - Q_{\mathbf{xu},t}Q_{\mathbf{uu},t}^{-1}Q_{\mathbf{u},t}\,,$$
$$V_{\mathbf{xx},t} = Q_{\mathbf{xx},t} - Q_{\mathbf{xu},t}Q_{\mathbf{uu},t}^{-1}Q_{\mathbf{ux},t}\,.$$

Here, similar to prior work, we use subscripts to denote derivatives. It can be shown (e.g., in (Tassa et al., 2012)) that the action $\mathbf{u}_t$ that minimizes the second-order approximation of the Q-function at every time step t is given by

$$\mathbf{u}_t = -Q_{\mathbf{uu},t}^{-1}Q_{\mathbf{ux},t}\mathbf{x}_t - Q_{\mathbf{uu},t}^{-1}Q_{\mathbf{u},t}\,.$$

This action is a linear function of the state $\mathbf{x}_t$, thus we can construct an optimal linear policy by setting $\mathbf{K}_t = -Q_{\mathbf{uu},t}^{-1}Q_{\mathbf{ux},t}$ and $\mathbf{k}_t = -Q_{\mathbf{uu},t}^{-1}Q_{\mathbf{u},t}$. We can also show that the maximum-entropy policy that minimizes the approximate Q-function is given by

$$p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, Q_{\mathbf{uu},t}).$$

This form is useful for LQR-FLM, as we use intermediate policies to generate samples to fit TVLG dynamics. Levine & Abbeel (2014) impose a constraint on the total KL-divergence between the old and new trajectory distributions induced by the policies through an augmented cost function $\bar{c}(\mathbf{x}_t,\mathbf{u}_t) = \frac{1}{\eta}c(\mathbf{x}_t,\mathbf{u}_t) - \log p^{(i-1)}(\mathbf{u}_t|\mathbf{x}_t)$, where solving for η via dual gradient descent can yield an exact solution to a KL-constrained LQR problem, where there is a single constraint that operates at the level of trajectory distributions $p(\tau)$. We can instead impose a separate KL-divergence constraint at each time step with the constrained optimization

$$\begin{aligned}\min_{\mathbf{u}_t,\boldsymbol{\Sigma}_t}\ &\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}_t),\mathbf{u}\sim\mathcal{N}(\mathbf{u}_t,\boldsymbol{\Sigma}_t)}[Q(\mathbf{x},\mathbf{u})]\\ s.t.\ &\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}_t)}[D_{KL}(\mathcal{N}(\mathbf{u}_t,\boldsymbol{\Sigma}_t)\|p^{(i-1)})] \le \epsilon_t\,.\end{aligned}$$

The new policy will be centered around $\mathbf{u}_t$ with covariance term $\boldsymbol{\Sigma}_t$. Let the old policy be parameterized by $\bar{\mathbf{K}}_t$, $\bar{\mathbf{k}}_t$, and $\bar{\mathbf{C}}_t$. We form the Lagrangian (dividing by η_t), approximate Q, and expand the KL-divergence term to get

$$\begin{aligned}&\mathcal{L}(\mathbf{u}_t,\boldsymbol{\Sigma}_t,\eta_t)\\ &= \frac{1}{\eta_t}\Big[Q_{\mathbf{x},t}^\top\mathbf{x}_t + Q_{\mathbf{u},t}^\top\mathbf{u}_t + \frac{1}{2}\mathbf{x}_t^\top Q_{\mathbf{xx},t}\mathbf{x}_t + \frac{1}{2}tr(Q_{\mathbf{xx},t}\Sigma_{\mathbf{x},t})\\ &\qquad + \frac{1}{2}\mathbf{u}_t^\top Q_{\mathbf{uu},t}\mathbf{u}_t + \frac{1}{2}tr(Q_{\mathbf{uu},t}\boldsymbol{\Sigma}_t) + \mathbf{x}_t^\top Q_{\mathbf{xu},t}\mathbf{u}_t\Big]\\ &\quad + \frac{1}{2}\Big[\log|\bar{\boldsymbol{\Sigma}}_t| - \log|\boldsymbol{\Sigma}_t| - d + tr(\bar{\boldsymbol{\Sigma}}_t^{-1}\boldsymbol{\Sigma}_t)\\ &\qquad + (\bar{\mathbf{K}}_t\mathbf{x}_t + \bar{\mathbf{k}}_t - \mathbf{u}_t)^\top\bar{\boldsymbol{\Sigma}}_t^{-1}(\bar{\mathbf{K}}_t\mathbf{x}_t + \bar{\mathbf{k}}_t - \mathbf{u}_t)\\ &\qquad + tr(\bar{\mathbf{K}}_t^\top\bar{\boldsymbol{\Sigma}}_t^{-1}\bar{\mathbf{K}}_t\Sigma_{\mathbf{x},t})\Big] - \epsilon_t\,.\end{aligned}$$

Now we set the derivative of $\mathcal{L}$ with respect to $\boldsymbol{\Sigma}_t$ equal to 0 and get

$$\boldsymbol{\Sigma}_t = \left(\frac{1}{\eta_t}Q_{\mathbf{uu},t} + \bar{\boldsymbol{\Sigma}}_t^{-1}\right)^{-1}\,.$$

Setting the derivative with respect to $\mathbf{u}_t$ equal to 0, we get

$$\mathbf{u}_t = -\boldsymbol{\Sigma}_t\left(\frac{1}{\eta_t}Q_{\mathbf{u},t} + \frac{1}{\eta_t}Q_{\mathbf{ux},t}\mathbf{x}_t - \hat{\mathbf{C}}_t^{-1}(\hat{\mathbf{K}}_t\mathbf{x}_t + \hat{\mathbf{k}}_t)\right)\,,$$

Thus our updated mean has the parameters

$$\mathbf{k}_t = -\boldsymbol{\Sigma}_t\left(\frac{1}{\eta_t}Q_{\mathbf{u},t} - \hat{\mathbf{C}}_t^{-1}\hat{\mathbf{k}}_t\right)\,,$$
$$\mathbf{K}_t = -\boldsymbol{\Sigma}_t\left(\frac{1}{\eta_t}Q_{\mathbf{ux},t} - \hat{\mathbf{C}}_t^{-1}\hat{\mathbf{K}}_t\right)\,.$$

As discussed by Tassa et al. (2012), when the updated $\mathbf{K}_t$ and $\mathbf{k}_t$ are not actually the optimal solution for the current quadratic Q-function, the update to the value function is a bit more complex, and is given by

$$V_{\mathbf{x},t} = Q_{\mathbf{x},t}^\top + Q_{\mathbf{u},t}^\top\mathbf{K}_t + \mathbf{k}_t^\top Q_{\mathbf{uu},t}\mathbf{K}_t + \mathbf{k}_t^\top Q_{\mathbf{ux},t}\,,$$
$$V_{\mathbf{xx},t} = Q_{\mathbf{xx},t} + \mathbf{K}_t^\top Q_{\mathbf{uu},t}\mathbf{K}_t + 2Q_{\mathbf{xu},t}\mathbf{K}_t\,.$$

8.2. PI^2 update through constrained optimization

The structure of the proof for the PI^2 update follows (Peters et al., 2010), applied to the cost-to-go $S(\mathbf{x}_t,\mathbf{u}_t)$. Let us first consider the cost-to-go $S(\mathbf{x}_t,\mathbf{u}_t)$ of a single trajectory or path $(\mathbf{x}_t,\mathbf{u}_t,\mathbf{x}_{t+1},\mathbf{u}_{t+1},\ldots,\mathbf{x}_T,\mathbf{u}_T)$ where T is the maximum number of time steps. We can rewrite the Lagrangian in a sample-based form as

$$\begin{aligned}&\mathcal{L}(p^{(i)},\eta_t) =\\ &\sum\left(p^{(i)}S(\mathbf{x}_t,\mathbf{u}_t)\right) + \eta_t\left(\sum p^{(i)}\log\frac{p^{(i)}}{p^{(i-1)}} - \epsilon\right)\,.\end{aligned}$$

Taking the derivative of $\mathcal{L}(p^{(i)},\eta_t)$ with respect to a single optimal policy $p^{(i)}$ and setting it to zero results in

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial p^{(i)}} &= S(\mathbf{x}_t,\mathbf{u}_t) + \eta_t\left(\log\frac{p^{(i)}}{p^{(i-1)}} + p^{(i)}\frac{p^{(i-1)}}{p^{(i)}}\frac{1}{p^{(i-1)}}\right)\\ &= S(\mathbf{x}_t,\mathbf{u}_t) + \eta_t\log\frac{p^{(i)}}{p^{(i-1)}} = 0\,.\end{aligned}$$

Solve the derivative for $p^{(i)}$ by exponentiating both sides

$$\log \frac{p^{(i)}}{p^{(i-1)}} = -\frac{1}{\eta_t} S(\mathbf{x}_t, \mathbf{u}_t)\,,$$

$$p^{(i)} = p^{(i-1)} \exp\left(-\frac{1}{\eta_t} S(\mathbf{x}_t, \mathbf{u}_t)\right)\,.$$

This gives us a probability update rule for a single sample that only considers cost-to-go of one path. However, when sampling from a stochastic policy $p^{(i-1)}(\mathbf{u}_t|\mathbf{x}_t)$, there are multiple paths that start in state $\mathbf{x}_t$ with action $\mathbf{u}_t$ and continue with a noisy policy afterwards. Hence, the updated policy $p^{(i)}(\mathbf{u}_t|\mathbf{x}_t)$ will incorporate all of these paths as

$$p^{(i)}(\mathbf{u}_t|\mathbf{x}_t) \propto p^{(i-1)}(\mathbf{u}_t|\mathbf{x}_t)\mathbb{E}_{p^{(i-1)}}\left[\exp\left(-\frac{1}{\eta_t} S(\mathbf{x}_t, \mathbf{u}_t)\right)\right]\,.$$

The updated policy is additionally subject to normalization, which corresponds to computing the normalized probabilities in Eq. (2).

8.3. Detailed Experimental Setup

8.3.1. Simulation Experiments

All of our cost functions use the following generic loss term on a vector $\mathbf{z}$

$$\ell(\mathbf{z}) = \frac{1}{2}\alpha\|\mathbf{z}\|_2^2 + \beta\sqrt{\gamma + \|\mathbf{z}\|_2^2}\,. \qquad (8)$$

α and β are hyperparameters that weight the squared ℓ_2 loss and Huber-style loss, respectively, and we set $\gamma = 10^{-5}$.

On the gripper pusher task, we have three such terms. The first sets $\mathbf{z}$ as the vector difference between the block and goal positions with $\alpha = 10$ and $\beta = 0.1$. $\mathbf{z}$ for the second measures the vector difference between the gripper and block positions, again with $\alpha = 10$ and $\beta = 0.1$, and the last loss term penalizes the magnitude of the fourth robot joint angle with $\alpha = 10$ and $\beta = 0$. We include this last term because, while the gripper moves in 3D, the block is constrained to a 2D plane and we thus want to encourage the gripper to also stay in this plane. These loss terms are weighted by 4, 1, and 1 respectively.

On the reacher task, our only loss term uses as $\mathbf{z}$ the vector difference between the arm end effector and the target, with $\alpha = 0$ and $\beta = 1$. For both the reacher and door opening tasks, we also include a small torque penalty term that penalizes unnecessary actuation and is typically several orders of magnitude smaller than the other loss terms.

On the door opening task, we use two loss terms. For the first, $\mathbf{z}$ measures the difference between the angle in radiants of the door hinge and the desired angle of -1.0, with $\alpha = 1$ and $\beta = 0$. The second is term is time-varying: for the first 25 time steps, $\mathbf{z}$ is the vector difference between the bottom of the robot end effector and a position above the door handle, and for the remaining time steps, $\mathbf{z}$ is the vector difference from the end effector to inside the handle. This encourages the policy to first navigate to a position above the handle, and then reach down with the hook to grasp the handle. Because we want to emphasize the second loss during the beginning of the trajectory and gradually switch to the first loss, we do a time-varying weighting between the loss terms. The weight of the second loss term is fixed to 1, but the weight of the first loss term at time step t is $5\left(\frac{t}{T}\right)^2$.

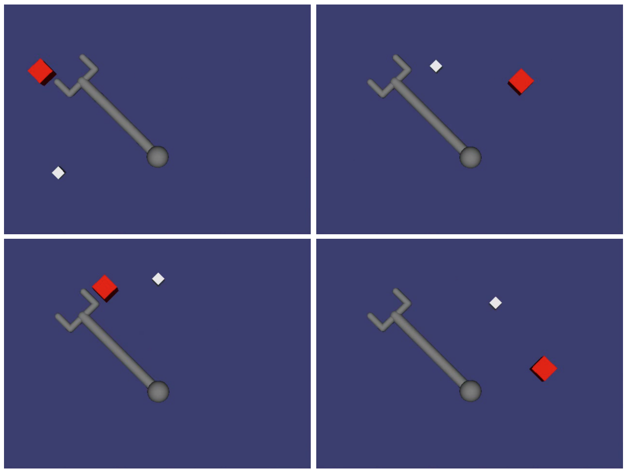
Figure 9. The initial conditions for the gripper pusher task that we train TVLG policies on. The top left and bottom right conditions are more difficult due to the distance from the block to the goal and the configuration of the arm. The top left condition results are reported in Section 6.1.

For the neural network policy architectures, we use two fully-connected hidden layers of rectified linear units (ReLUs) with no output non-linearity. On the reacher task, the hidden layer size is 32 units per layer, and on the door opening task, the hidden layer size is 100 units per layer.

All of the tasks involve varying conditions for which we train one TVLG policy per condition and, for reacher and door opening, train a neural network policy to generalize across all conditions. For gripper pusher, the conditions vary the starting positions of the block and the goal, which can have a drastic effect on the difficulty of the task. Figure 9 illustrates the four initial conditions of the gripper pusher task for which we train TVLG policies. For reacher, analogous to OpenAI Gym, we vary the initial arm configuration and position of the target and train TVLG policies from 16 randomly chosen conditions. Note that, while OpenAI Gym randomizes this initialization per episode, we always reset to the same condition when training TVLG policies as this is an additional assumption we impose. However, when we test the performance of the neural network policy, we collect 300 test episodes with random initial conditions. For the door opening task, we initialize the robot position within a small square in the ground plane. We train TVLG policies from the four corners of this square and test our neural network policies with 100 test episodes from random positions within the square.

For the gripper pusher and door opening tasks, we train TVLG policies with PILQR, LQR-FLM and PI^2 with 20 episodes per iteration per condition for 20 iterations. In Appendix 8.4, we also test PI^2 with 200 episodes per iteration. For the reacher task, we use 3 episodes per iteration per condition for 10 iterations. Note that we do not collect any additional samples for training neural network policies. For the prior methods, we train DDPG with 150 episodes per epoch for 80 epochs on the reacher task, and TRPO uses 600 episodes per iteration for 120 iterations. On door opening, TRPO uses 400 episodes per iteration for 80 iterations and DDPG uses 160 episodes per epoch for 100 epochs, though note that DDPG is ultimately not successful.

8.3.2. Real Robot Experiments

For the real robot tasks we use a hybrid cost function that includes two loss terms of the form of Eq. 8. The first loss term $\ell_{arm}(\mathbf{z})$ computes the difference between the current position of the robot's end-effector and the position of the end-effector when the hockey stick is located just in front of the puck. We set $\alpha = 0.1$ and $\beta = 0.0001$ for this cost function. The second loss term $\ell_{goal}(\mathbf{z})$ is based on the distance between the puck and the goal that we estimate using a motion capture system. We set $\alpha = 0.0$ and $\beta = 1.0$. Both ℓ_{arm} and ℓ_{goal} have a linear ramp, which makes the cost increase towards the end of the trajectory. In addition, We include a small torque cost term ℓ_{torque} to penalize unnecessary high torques. The combined function sums over all the cost terms: $\ell_{total} = 100.0\,\ell_{goal} + \ell_{arm} + \ell_{torque}$. We give a substantially higher weight to the cost on the distance to the goal to achieve a higher precision of the task execution.

Our neural network policy includes two fully-connected hidden layers of rectified linear units (ReLUs). Each of the hidden layers consists of 42 neurons. The inputs of the policy include: puck and goal positions measured with a motion capture system, robot joint angles, joint velocities, the end-effector pose and the end-effector velocity. During PILQR-MDGPS training, we use data augmentation to regularize the neural network. In particular, the observations were augmented with Gaussian noise to mitigate overfitting to the noisy sensor data.

8.4. Additional Simulation Results

Figure 10 shows additional simulation results obtained for the gripper pusher task for the three additional initial conditions. The instances presented here are not as challenging as the one reported in the paper. Our method (PILQR) is

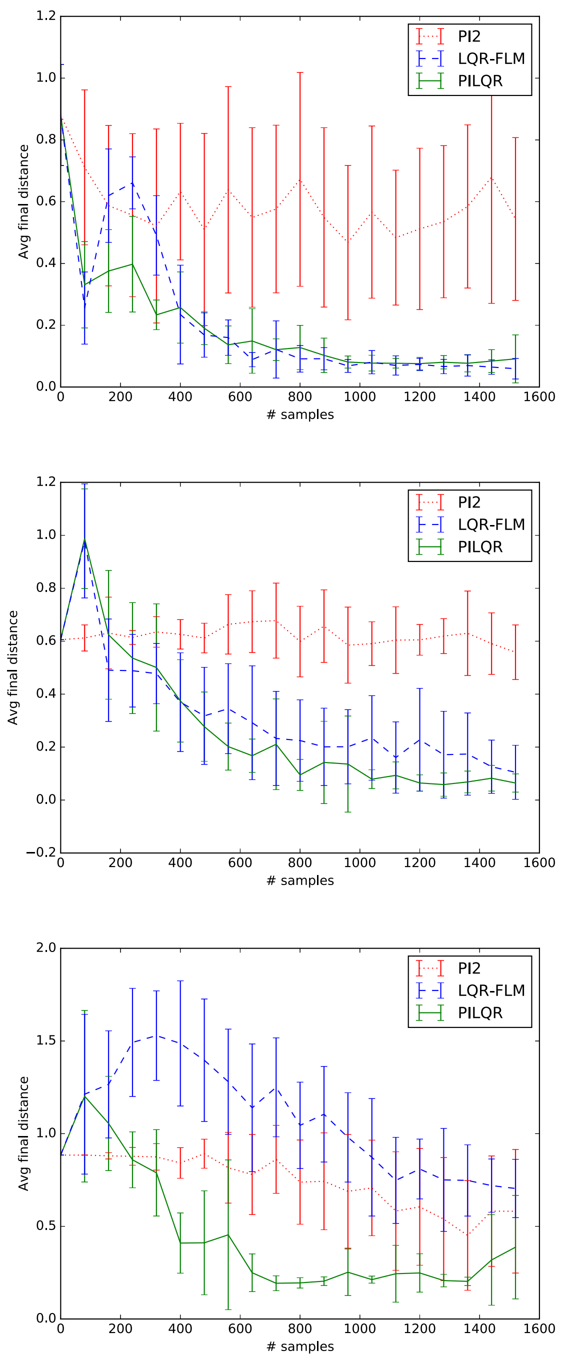

Figure 10. Single condition comparisons of the gripper-pusher task in three additional conditions. The top, middle, and bottom plots correspond to the top right, bottom right, and bottom left conditions depicted in Figure 9, respectively. The PILQR method outperforms other baselines in two out of the three conditions. The conditions presented in the top and middle figure are significantly easier than the other conditions presented in the paper.

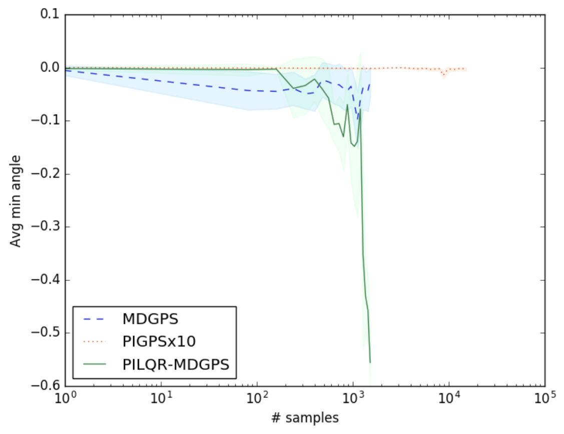

Figure 11. Additional results on the door opening task.

able to outperform other baselines except for the first two conditions presented in the first rows of Figure 10, where LQR-FLM performs equally well due to the simplicity of the task. PI^2 is not able to make progress with the same number of samples, however, its performance on each condition is comparable to LQR-FLM when provided with 10 times more samples.

We also test PI^2 with 10 times more samples on the reacher and door opening tasks. On the reacher task, PI^2 improves substantially with more samples, though it is still worse than the four other methods. However, as Figure 11 shows, PI^2 is unable to succeed on the door opening task even with 10 times more samples. The performance of PI^2 is likely to continue increasing as we provide even more samples.