Online Experiment Design for System Identification of LTI Systems using Reinforcement Learning

Eugene Vinitsky* , Zian Hu†, Alexandre Bayen†‡§
*UC Berkeley, Department of Mechanical Engineering
†UC Berkeley, Department of Mathematics
‡UC Berkeley, Department of Electric Engineering and Computer Science
§UC Berkeley, Institute for Transportation Studies

Abstract— **Using policy gradient methods, we investigate the problem of learning a controller that can construct online optimal experiment designs for estimation of controllable, linear time-invariant systems. We train a controller to condition its choice of input on the prior history of observed input-output pairs; we then feed the input and output pairs to least squares to estimate the dynamics matrices. We show that the trained controller can pick out samples that lead to several orders of magnitude improvements in estimation accuracy as compared against a random Gaussian baseline and can be used to synthesize stable LQR controllers with a higher success rate than a robust synthesis procedure and has significantly lower LQR cost. We demonstrate empirically that the use of all the samples of the trajectory only leads to a marginal improvement in the accuracy of the estimations; the controller is able to estimate systems almost perfectly with a budget of six input samples. Furthermore, we demonstrate that our system is able to accurately estimate both stable and unstable LTI systems. Finally, we open-source the trained policies to enable benchmarking of other optimal input design algorithms against the controller.**

I. Introduction

The problem of discovering the optimal input to estimate an unknown dynamical system is a fundamental question stemming back to the 70's [1]. For dynamical systems for which acquiring samples can be expensive or risky such as medical systems or chemical plants, it is essential to be able to discover the dynamics of the system with maximal efficiency. However, synthesizing the optimal input for systems with a large number of unknown parameters can be computationally expensive or intractable.

In recent years, deep reinforcement learning (RL), the combined use of reinforcement learning algorithms with neural networks as the controllers, has proven to be competitive on a wide range of control tasks ranging from four legged robot recovery from falls [2], robotic grasping [3], to trajectory tracking [4]. Intriguingly, RL has also turned out to be useful in algorithm design, being used to learn new gradient descent update rules [5], generate unconstrained optimization algorithms [6], and to output solutions to combinatorial optimization problems [7].

In the present work we aim to combine the two concepts, applying deep RL to the classic problem of optimal experiment design for system identification (sys-ID). Whereas the experiment design literature appears to have primarily focused on offline synthesis of the optimal input, in this work we will focus on *online experiment design*, in which the newly synthesized input is a function of the prior observed input-output pairs. By taking into account prior observations, we can focus the next set of inputs on model parameters that are poorly estimated or more uncertain.

This work is, to our knowledge, the first to apply learning to tackle the question of how to optimally select samples to be fed to a least-squares estimation procedure. The contributions of this article are as follows:

- Using reinforcement learning, train a controller that takes into account all prior observed input-output pairs when selecting the next input.
- Demonstrate that the controller outperforms a random Gaussian baseline by four orders of magnitude in estimating the elements of the dynamics matrices.
- Empirically demonstrate that the controller can be used to find stabilizing feedback controllers with higher probability than a robust synthesis procedure.

The rest of the article is organized as follows:

- In Sec. II we describe prior work on optimal experiment design and closely related work on optimal design for the LQR problem.
- In Sec. III we briefly describe experiment design, LTI systems, the *Linear Quadratic Regulator*, Reinforcement Learning, and policy gradient methods and provide an introduction to the robust synthesis benchmark we use.
- In Sec. IV we provide details on the experiments we ran.
- In Sec. V and Sec. VI we explain the results of our experiment, discuss whether the controller generalizes, and compare against the available baselines.

II. Related Work

The closest area of research to the problem considered here is the adaptive/sequential input design literature in which the optimal input is reassessed after each experiment. To solve the standard problem that the optimal input is usually a function of the parameters of the unknown system, in [8], the authors alternately perform sys-ID and use the estimate of the new system to compute the solution of a convex problem

Corresponding author: Eugene Vinitsky (evinitsky@berkeley.edu)
Email addresses:{zian.hu, evinitsky, bayen}@berkeley.edu

that outputs an optimal input. This procedure however, does not provably converge. Although this procedure performed well on a single provided example, its overall efficacy relative to the optimal input is unclear. [9] builds on this prior work to develop asymptotic convergence criteria for an adaptive procedure for estimating stable, linear autoregressive models but only provides convergence guarantees for a restricted class of inputs. Finally, in [10], the authors develop an online input experiment design procedure for single-input single-output linear time-invariant systems that has the same asymptotic behavior as an off-line estimator that has access to the system parameters necessary to design an optimal experiment design. However, their procedure is limited to stable LTI systems.

There is also related work on general online estimation problems of LTI systems. In [11] they skip the model estimation step and directly try to estimate the H_∞ norm of the underlying system. In contrast to our work here, they find that for norm estimation problem the state of the art adaptive sampling procedures do not outperform a non-adaptive sampling procedure.

III. Background

A. Optimal Experiment Design

The optimal experiment design problem, as applied to dynamical systems is to find an input $u(t)$ that, according to some optimality criterion, minimizes the error between the true model $M(\theta_0)$ and the model $M(\theta)$ reconstructed after performing system identification on the observed input-output pairs. In early work on optimal experiment design such as the DETMAX algorithm [12] and [13], the main object of interest is the co-variance matrix of the estimator of θ, P_θ. This work spawned a large number of optimality criteria such as A-optimality (minimize $tr(P_\theta)$), D-optimality (minimize $\det(P_\theta)$, L-optimality (minimize λ_{max}) as well as analytical solutions for the optimal input. Unfortunately, these analytical solutions tend to be functions of the unknown parameter system and consequently require the construction of additional assumptions on the model parameters such as being contained in some set or distributed around a nominal value [14]. More recent work has focused on extending optimal experiment design to increasingly complex optimality criteria, characterizing the trade-off between experiment complexity and subsequent controller synthesis, and closed loop optimal experiment design. For a more extensive history and survey of optimal experiment design formulations, one can look at [14] [13].

B. Linear Time Invariant, Controllable Systems

We briefly characterize controllability in linear time-invariant (LTI), fully-observed, discrete systems. The dynamics of the system studied in this work are given by

$$x_{t+1} = Ax_t + Bu_t + w_t \tag{1}$$

where for the purpose of simplicity $A,\ B \in \mathbb{R}^{n\times n}$, $x_t,\ u_t \in \mathbb{R}^n$, and in the case of this article, $w_t \sim \mathcal{N}(0, \sigma^2 I)$. To check controllability of the system we use the standard rank test, checking that the matrix

$$C = \left[B, AB, A^2B, \dots, A^{n-1}B\right] \tag{2}$$

has full rank, i.e. rank $(C) = n$.

C. Linear Quadratic Regulator

To encourage our system to not only output accurate estimates of A and B but also to prioritize outputting estimates $\hat{A},\ \hat{B}$ that maintain the controllability property of A and B, we use a quadratic cost function on the states x_t and actions u_t as described in more detail in Sec. IV-B. In particular, for each estimated A and B, which we refer to as $\hat{A}$ and $\hat{B}$, we evaluate the reward with respect to the finite horizon *Linear Quadratic Regulator* (LQR) problem:

$$\begin{aligned} J = &\min_{u_0,\dots,u_{N-1}} \mathbb{E}_{x_0,w}\left[\sum_{t=0}^{N-1} x_t^T Q x_t + u_t^T R u_t + x_N^T Q x_N\right] \\ &\text{subject to } x_{t+1} = Ax_t + Bu_t + w_t \end{aligned} \tag{3}$$

where $x_t,\ u_t,\ w_t$ are the state, action, and noise respectively. $Q \succ 0$ and $R \succ 0$ are weighting matrices on the state and input. Here the expectation is taken over a chosen distribution of noise and starting position x_0. For evaluation we will also use the infinite horizon LQR problem:

$$\begin{aligned} J = \lim_{N\to\infty} &\min_{u_0,\dots,u_{N-1}} \frac{1}{N}\mathbb{E}_{x_0,w}\left[\sum_{t=0}^{N-1} x_t^T Q x_t + u_t^T R u_t\right] \\ &\text{subject to } x_{t+1} = Ax_t + Bu_t + w_t \end{aligned} \tag{4}$$

We note that we have a simple analytical solution to the infinite horizon stochastic LQR problem under the following assumptions:

- The noise terms w_t are independent and identically distributed.
- $\mathbb{E}\left[w_t\right] = 0$, $\mathbb{E}\left[w_t w_t^T\right] = W$.
- x_0 is independent of w_t, $\mathbb{E}\left[x_0\right] = 0$, $\mathbb{E}\left[x_0 x_0^T\right] = X$.
- The pair (A, B) is controllable and $Q \succ 0,\ R \succ 0$.

If the above conditions are satisfied then the system has the analytic solution $J^* = \boldsymbol{Tr}\left(WP\right)$ where P is the solution to the discrete-time algebraic Ricatti equation [15]

$$P = A^T PA - (A^T PB)(R + B^T PB)^{-1}(B^T PA) + Q \tag{5}$$

We use this analytical solution extensively in constructing the plots in Sec. V.

D. Least Squares Solution of LTI Systems

We briefly describe the least squares solution used to compute our estimates of A and B, which we refer to as $\hat{A}$ and $\hat{B}$. We run one rollout by tracking N trajectories of the system with initial states of $x_0 = \mathbf{0}^n$, and collect state, action pairs for every step. Given a rollout consisting of N trials each of length T, we obtain the dataset $\{(x_t^j, u_t^j) : 0 \leq$

$t \leq T, 1 \leq j \leq N\}$ where $x_0^j = \mathbf{0}^n$ for all j. The estimates $\hat{A}$ and $\hat{B}$ are then solutions to the least squares matrix problem

$$(\hat{A}, \hat{B}) \in \arg\min_{(A,B)} \sum_{j=1}^{N} \sum_{t=0}^{T-1} \frac{1}{2} ||Ax_t^j + Bu_t^j - x_{t+1}^j||_2^2 \quad (6)$$

To derive the analytic expression for the least squares estimator, we follow the same procedure as outlined in [13]. To summarize, we begin by defining $\Theta := [A, B]^*$, where $\Theta \in \mathbb{R}^{(2n \times n)}$. Additionally, let $z_t := \begin{bmatrix} x_t \\ u_t \end{bmatrix} \in \mathbb{R}^{2n}$ Using this notation, we obtain after a single rollout

$$X := \begin{bmatrix} x_1^{1*} \\ \vdots \\ x_T^{1*} \\ \vdots \\ x_1^{N*} \\ \vdots \\ x_T^{N*} \end{bmatrix} Z := \begin{bmatrix} z_0^{1*} \\ \vdots \\ z_{T-1}^{1*} \\ \vdots \\ z_0^{N*} \\ \vdots \\ z_{T-1}^{N*} \end{bmatrix} W := \begin{bmatrix} w_0^{1*} \\ \vdots \\ w_{T-1}^{1*} \\ \vdots \\ w_0^{N*} \\ \vdots \\ w_{T-1}^{N*} \end{bmatrix} \quad (7)$$

where each expression consists of all data from each of the N trials stacked vertically. We can thus write $X = Z\Theta + W$, implying that our least squares estimator $\hat{\Theta}$ can be written as $\hat{\Theta} = (Z^*Z)^{-1}Z^*X$, provided that Z^*Z is invertible.

E. Deep Reinforcement Learning and MDPs

In this section, we discuss the notation and describe in brief the key ideas used in reinforcement learning. Reinforcement learning focuses on maximization of the discounted reward of a finite-horizon *Markov decision process* (MDP) [16]. The system described in this article solves tasks which conform to the standard structure of a finite-horizon discounted MDP, defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma, T)$, where $\mathcal{S}$ is a (possibly infinite) set of states, $\mathcal{A}$ is a set of actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ is the transition probability distribution for moving from one state s to another state s' given action a, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}_{\geq 0}$ is the initial state distribution, $\gamma \in (0, 1]$ is a discount factor, and $T : \mathcal{S} \times \mathcal{A} \to 0, 1$ is a termination condition. When T only maps to one after a fixed number of time-steps and is zero otherwise, it is referred to as a *horizon*. We will refer to a series of state, action, reward pairs $(s_0, r_0, a_0 \ldots, a_{t-1}, s_t, r_t)$ from 0 until the termination condition is hit (in this tuple it is hit at time t) as a *rollout*. We will also interchangeably use the term *rollout* to refer to running the system until the termination condition is hit.

RL studies the problem of how an agent can learn to take actions in its environment to maximize its expected cumulative discounted reward: specifically it tries to maximize $J^\pi = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t \middle| \pi(a_t|s_t)\right]$ where r_t is the reward at time t and $\pi : \mathcal{S} \to \mathcal{A}$, is a controller mapping states to actions. When π is a probability distribution over actions it is written as $\pi(a|s)$. It is increasingly popular to parametrize the controller via a neural net. We will denote the parameters of this controller, also known as neural network weights, by θ and the controller parametrized by θ as π_θ. A neural net consists of a stacked set of affine linear transforms and nonlinearities. The presence of many stacked layers is the origin of the term "deep" reinforcement learning.

F. Policy Gradient Methods: Proximal Policy Optimization

While there are many algorithms for finding the optimal policy of the expected discounted reward J^π, an effective method is to estimate the gradient of J^π with respect to the policy parameters using Monte Carlo sampling [17] and then perform stochastic gradient ascent: these methods are referred to as policy gradient algorithms. In this work we use *proximal policy optimization* (PPO), a policy gradient algorithm that clips the loss function if the gradient step would cause too drastic a shift in the underlying policy. PPO enables taking multiple gradient steps per batch of trajectories and has demonstrated improved sample complexity (number of trajectories to reach a good reward) on a wide range of common RL tasks. For more details please see [18].

G. Robust Synthesis Benchmark

Due to a lack of available benchmarks in online optimal experiment design, we instead choose to benchmark against a robust synthesis procedure for feedback controllers [19]. In this work, the authors use the *System Level Synthesis* [20] procedure to construct an optimization problem whose output is a transfer function parametrization of the stabilizing feedback controller with guaranteed bounds on its sub-optimality in terms of bounds on the error in bootstrapped, least-squares estimates of A and B. While the resulting optimization problem is intractable, they develop a program that can closely approximate the result by only expressing the transfer function out to a finite number of terms. In Sec. IV the results of this optimization procedure are depicted in Fig. 2 and Fig. 3 as the lines marked *FIR*.

We note that using this work as a benchmark is not a perfect comparison, as their work is intended to provide formal guarantees of the feasibility and sub-optimality of finding a stabilizing feedback controller, rather than find a stabilizing feedback controller with as high a rate of success as possible. We primarily use this work to point out how beneficial input design can be in constructing good estimates. However, as discussed in VI, their approach is complementary to ours and the two methods can be combined for an even more effective synthesis procedure.

IV. Experiments

A. Design Considerations

We seek to develop a controller that can pick out inputs that will lead to estimates $\hat{A}$, $\hat{B}$ that have small error relative to the true dynamics, A, B. We measure the error between these matrices as $\epsilon_A = ||\hat{A} - A||_2 = \sigma_{\max}(\hat{A} - A)$ where $\sigma_{\max}$ is the largest singular value. We want our controller to be able to sample optimal inputs given a potentially arbitrary number of samples, so, during the training we vary how many trajectories the controller is allowed to control before the input-output pairs are fed to the least-squares estimator. To ensure that the controller is aware of its variable budget, the number of allotted trials and the length of each trial will be fed to the controller as input. We denote the number of trials by $N \in [N_{\text{min}}, N_{\text{max}}]$ and the length of the trials as L. Because the controller must take in a fixed number of inputs, we set a maximum number of possible trials as N_{max} and replace the missing input-output pairs with zeros.

B. MDP design

To define the MDP we define the state space, action space, reward function, and termination condition. To enable the controller to adaptively account for previous observed input-output pairs, s_t consists of all the observed input and output pairs, padded with zeros to ensure a fixed length and then appended with the trial length and number of trials alloted in that rollout. In particular, a given state $\mathcal{S} \in \mathbb{R}^{2nLN_{\text{max}}+2}$ will consist of:

$$s_t = \Big[x_0, \ldots, x_{t-1}, \mathbf{0}^{n(LN_{\text{max}}-t)}, u_0, \ldots, u_{t-1}, \\ \mathbf{0}^{n(LN_{\text{max}}-t)}, L, N\Big] \tag{8}$$

Here $\mathbf{0}^{n(LN_{\text{max}}-t)}$ is a zero padding vector ensuring that the input to the neural network is always of fixed length. The action is simply a vector of appropriate dimension for the system, $\mathcal{A} \in R^n$. Since we benchmark against a system [19] where the inputs are drawn from the Gaussian $\mathcal{N}(0, I)$, we bound the magnitude of the controller actions to be between -1 and 1 so as to allow for a fair comparison: $a \in [-1, 1]$.

For the reward function, we want to prevent the controller from trading off accuracy in estimating A and B, so rather than use a reward around the distance between $\hat{A}$, $\hat{B}$ and A, B, we use the estimated $\hat{A}, \hat{B}$ to synthesize the optimal estimated solution to the Ricatti equation, $\hat{K}$ and evaluate the LQR cost of $J(\hat{K})$ versus the optimal cost $J(K^*)$. To prevent the reward from going off to infinity if the closed loop system is unstable, we simply step the closed loop system forwards L steps and compute the distance to the optimal cost $J(K^*) - J(\hat{K})$ over those L steps. During the sampling portion of the control, before the least squares estimation, we do not have estimates so we simply return the reward as zero. Formally, this yields:

$$r_t = \begin{cases} 0, & t < NL \\ \max(J_L(K) - J_L(\hat{K}), -r_{\text{penalty}}) & t = NL \end{cases} \tag{9}$$

where $J_L(K)$ is the empirical LQR cost using controller K for L steps and where the initial state $x_0 \sim \mathcal{N}(\mathbf{0}^n, I^{n \times n})$. Finally, $-r_{\text{penalty}}$ is a penalty used to clip the reward if it gets too negative.

For the termination condition, at each rollout we sample $N \sim \text{unif}(N_{\text{min}}, N_{\text{max}})$. Once we hit $t = NL$ samples, the rollout ends and the next rollout begins.

A and B are defined as follows: at the start of each rollout we randomly sample all the elements A and B from $\text{unif}(-\text{e}_{\text{max}}, \text{e}_{\text{max}})$ where e_{max} represents bounds on the largest possible elements in A and B. As discussed in Sec. IV-D, this corresponds to a bound on the top eigenvalues of A and B. If the sampled A and B pair do not pass the controllability condition given in Sec. III-B, we simply sample A and B repeatedly until an A, B pair is found for which the controllability condition is met.

The dynamics of the system then proceed as the regular LTI dynamics $x_{t+1} = Ax_t + Bu_t + w_t$ with one exception: every L steps we reset the state x_t of the system back to zero. This is to provide for easier comparison with existing benchmarks.

C. Algorithm/controller details

For our controller $\pi(a|S)$, we use a Diagonal Gaussian parametrization. The controller takes in a state s_t, passes it through the neural network, and outputs a mean and a diagonal-covariance matrix μ, Σ. The action is then sampled as $a \sim \mathcal{N}(\mu, \Sigma)$. This parametrization is fairly standard in RL as it allows for exploration due to the randomness, but can converge on a relatively deterministic policy by shrinking the magnitude of the covariance matrix. For the neural network, we used hidden dimensions of [256, 256, 256], and a $\tanh$ non-linearity.

For the training parameters for PPO we used the default parameters set in RLlib [21][1] version 0.6.3, a distributed deep RL library, with the exception of performing a hyperparameter sweep over the stochastic gradient descent stepsize (learning-rate). The best performing value, in terms of final converged reward, was used. The final learning rates chosen were $1e^{-3}$ and $8.5{*}1e^{-4}$ for the full least-squares experiment and partial least-squares experiment respectively. We used a training batch size of 30000 total steps. Finally, to train the system we used m4.10xlarge Amazon Web server instances and parallelized the training over 38 CPUS. Using these computational resources, the experiments converged in 12 hours.

In the spirit of reproducibility, all code used to train the controllers, the controllers themselves, as well as the code used to generate the graphs in this article are available at our Github repository[2]. To re-run the experiments, use the trained controllers, and recreate the graphs, please visit the page and read the tutorials.

D. Experiments run

We run two sets of experiments:

- (1) An experiment where $n = 3$ and we use all of the input-output pairs from the rollout as input to the least squares. We refer to this as the *full least-squares experiment*.
- (2) An experiment where $n = 3$ and we use only the last input-output pairs as input to the least squares. The MDP is unchanged from the prior experiment, the only change is in what the least squares receives. We refer to this as the *partial least-squares experiment*.

Since we will compare against a benchmark that uses only the last input-output pair of each of its rollouts, the second experiment is used to confirm that the improvement caused by experiment design is not solely due to an increase in the

[1]https://github.com/ray-project/ray/python/ray/rllib
[2]https://github.com/zianhu7/LQR

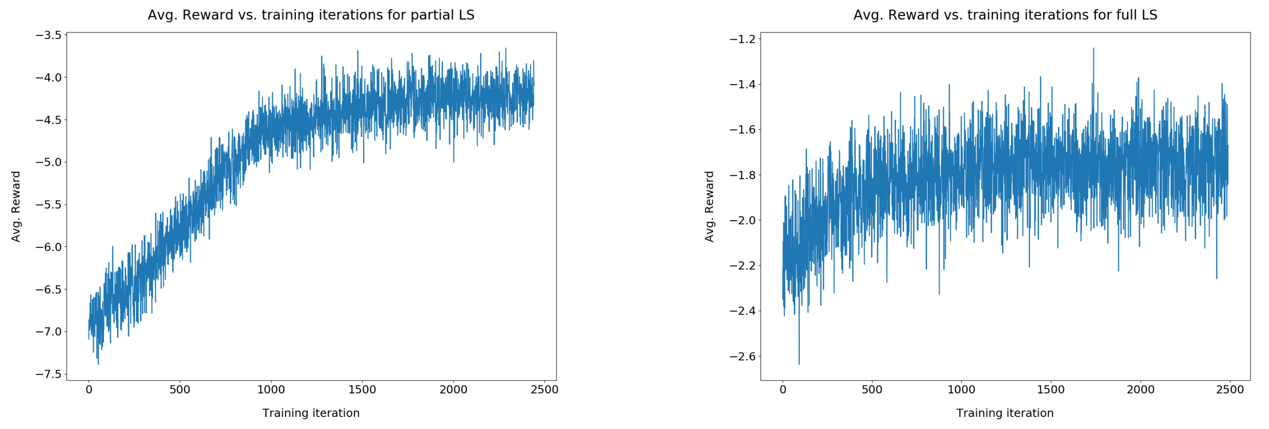

Fig. 1: (Top): Training curve for the partial least squares experiment. The training has visibly converged at a value around -4.0. (Bottom): Training curve for the full least squares experiment. The training has visibly converged at a value around -1.5. Note that the average reward is higher than the partial case.

number of observed samples but is also due to intelligently selecting the inputs.

For the values above we use:

- Number of trials: N_{min} = 6, N_{max} = 20.
- System dimension: $n = 3$.
- Length of a trial: $L = 6$.
- $Q = 10^{-3}\boldsymbol{I^{n\times n}},\ R = \boldsymbol{I^{n\times n}}$.
- the noise is: $w_t \sim \mathcal{N}\left(0, \boldsymbol{I^{n\times n}}\right)$.

We also constrain the top eigenvalues of the system by bounding the elements of both A and B to be in $[-2, 2]$. By the Gershgorin Disc Theorem [22], this bounds the eigenvalues of the system λ to have bounded magnitude $|\lambda| < 6$. We did this under the assumption that bounding the eigenvalues would make the learning problem easier as the reward would hit the penalty value less often.

Finally, for the evaluation where we compare with the robust synthesis procedure from Sec. III-G, we use the following dynamic system for the evaluations depicted in Fig. 2 and Fig. 3:

$$x_{t+1} = \begin{bmatrix} 1.01 & 0.01 & 0 \\ 0.01 & 1.01 & 0.01 \\ 0 & 0.01 & 1.01 \end{bmatrix} x_t + I^{3\times 3} u_t, \qquad (10)$$
$$x_t, u_t \in \mathbb{R}^3$$

V. Results

Fig. 1 depicts the average reward for the partial and full least squares. The curve in both cases has leveled out, indicating that the controller has converged. As might be expected, the full least squares case has a significantly higher reward once converged.

Each of the following plots, with the exception of the robust synthesis curves, is constructed by running the sampling procedure for $5*10^5$ total steps and recording the results. As is done during training time, the allowed number of total trajectories is randomly sampled from $N \sim \text{unif}\left(N_{\text{min}}, N_{\text{max}}\right)$. For the curves marked *Gaussian*, we simply sampled from $\mathcal{N}\left(0, \boldsymbol{I}^{3\times 3}\right)$ to get the inputs. For the *FIR* curves corresponding to the robust synthesis procedure, the data-points are extracted from [19] using the *WebPlotDigitizer*[3] tool.

In Fig. 2 we depict the percentage of time that the closed loop system, $A + B\hat{K}$ where K is computed from the Ricatti equation solved using $\hat{A}, \hat{B}$, is stable for the dynamics given in Eq. 10. For the same set of runs, we also depict in Fig. 3 the median LQR relative cost suboptimality, defined as $\frac{|J(K)-J(\hat{K})|}{J(K)}$, between the estimated $\hat{K}$ and the true K. If the closed loop system is unstable, we replace the infinite cost suboptimality with 10^6 for visual purposes.

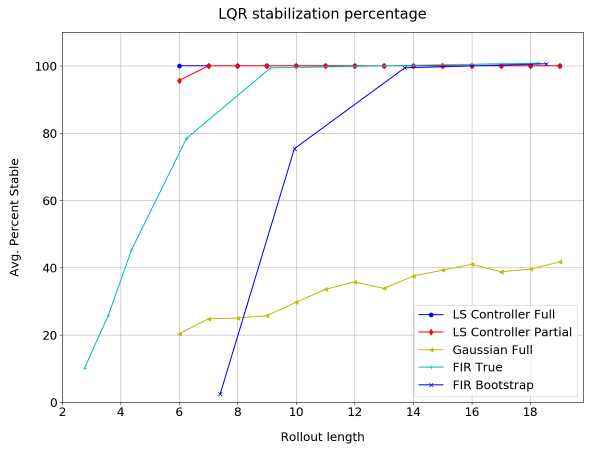

Fig. 2: Comparison of the percentage of found stabilizing controllers as a function of the number of trajectories used to generate samples.

Since we trained under a restricted range of eigenvalues for $|\lambda_i(A)| < 6,\ |\lambda_i(B)| < 6\ \forall i$, an interesting question is whether the trained controller has simply memorized the range of possible inputs or learned a general procedure. To

[3]https://automeris.io/WebPlotDigitizer/

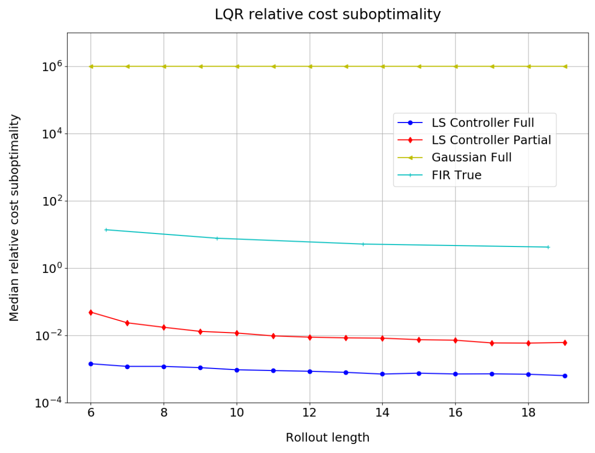

Fig. 3: Comparison of the median relative cost of the different approaches. Sampling with a Gaussian (yellow) is unstable and thus achieves the maximum cost of 10^6, the partial input (red) and full input (blue) outperform the robust synthesis (cyan).

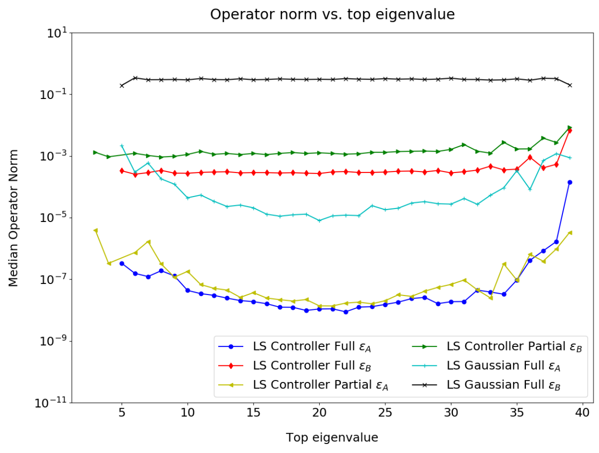

Fig. 5: Operator norm of $||A - \hat{A}_2||$ and $||B - \hat{B}||_2$ for Gaussian, partial, and full least squares control. Error in A is lower than error in B as might be expected. The input design is 3 orders of magnitude better than random sampling.

investigate this, we randomly sample 4000 A and B matrices with eigenvalues between 0 and 40 and depict the results in Fig. 4.

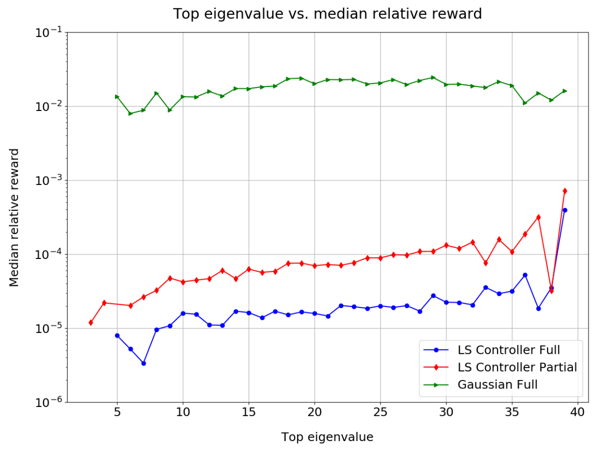

Fig. 4: Test of generalization. Trained controllers with both full and partial observation of input-output pairs (red, blue) outperform the Gaussian and perform almost equally well outside of their training range of top eigenvalue of 6.

Finally, we investigate just how accurate the estimates $\hat{A}, \hat{B}$ are by randomly sampling approximately 4000 A and B matrices with eigenvalues from 0 to 40 and computing $\epsilon_A = ||\hat{A} - A||_2, \ \epsilon_B = ||\hat{B} - B||_2$ as a function of the top eigenvalue. A comparison of the results against a random Gaussian baseline is depicted in Fig. 5.

VI. Discussion

As can be seen from Fig. 2, both the partial and full least squares controllers almost immediately stabilize the example system in Eq. 10 at the minimal number of trajectories. Furthermore, the trained controller sharply outperforms the results of performing least squares on inputs sampled from a Gaussian, so the controller has clearly learned some advantageous behavior over pure randomness. Interestingly, both the partial and full controllers outperform the robust synthesis procedure. Since the partial controller uses the same number of input samples as the robust synthesis procedure, there must be some improvement coming from intelligent input selection that equals the effect of robustly computing the feedback controller.

We can understand the reason for the improvement further by examining Fig. 5. Here we can see that the controller produces estimates of A and B that are four orders of magnitude lower than sampling from a Gaussian to get the inputs. Since the robust synthesis procedure uses a Gaussian input to compute $\hat{A}$, $\hat{B}$, it is likely that our highly accurate estimates of A and B are a part of the reason for the improvement. As noted in [23], the difference in LQR cost between the optimal controller and a least-squares estimated controller scales quadratically with the model error ϵ where $||\hat{A} - A||_2 < \epsilon, \ ||\hat{B} - B||_2 < \epsilon$ whereas the robust controller in [19] scales linearly in ϵ. Thus, the four orders of magnitude in LQR cost scales perfectly with the four orders of magnitude improvement in estimation of the elements of $A, \ B$ over the Gaussian baseline.

We can also see from Fig. 4 and Fig. 5 that the controller is still giving improved estimates for systems whose top eigenvalue is a good deal larger than the top training eigenvalue of 6. This suggests that the controller has learned an input experiment design procedure that generalizes outside of the training distribution and may have learned a procedure that is applicable to arbitrary controllable A, B matrices. While the performance does appear to decrease slightly at larger eigenvalues in Fig. 5, the decrease also occurs in the

Gaussian estimate, suggesting that these higher eigenvalue systems may just be harder to stably estimate.

VII. Conclusions and Future Work

In this work we studied the problem of online experiment design and learned a controller that can take a prior history of observed states and inputs and use it to pick out inputs that improve the performance of the least squares sys-ID procedure for LTI systems. This controller can be plugged in as a component of any system that uses least squares to estimate its dynamics. We also demonstrated that the controller generalizes to high eigenvalue systems. Furthermore, we showed that the system outperforms sampling the inputs from a Gaussian by four orders of magnitude. Finally, we showed that the improvement in accuracy of the dynamics estimate is enough to cause our system to outperform a robust synthesis procedure at the cost of giving up formal guarantees on the performance.

An interesting line of inquiry is finding places where this controller can serve as the input to other procedures, for example, would it improve the performance of the robust synthesis procedure it was benchmarked against? Could it be used for online control of non-linear dynamical systems by rapidly computing local linear approximations? A parallel thread of future inquiry is to investigate whether this approach of learning an online experiment designer scales to systems beyond just linear, time-invariant ones. It is unlikely that linear systems are the sole domains where learning controllers for online experiment design could be effective.

Finally, our controller currently does not come with any theoretical guarantees of performance. However, all of the actions are sampled from a discrete series of Gaussians. Since works like [19] are able to develop guarantees on the estimation performance under the assumption of sampling from a constant Gaussian, it is likely possible to develop guarantees for the expected performance of the controller.

VIII. Acknowledgements

Computational resources for this work were provided by an AWS Machine Learning grant. Eugene Vinitsky is supported by an NSF Graduate research fellowship.

References

[1] M. Aoki and R. Staley, "On input signal synthesis in parameter identification," *Automatica*, vol. 6, no. 3, pp. 431–440, 1970.

[2] J. Lee, J. Hwangbo, and M. Hutter, "Robust recovery controller for a quadrupedal robot using deep reinforcement learning," *arXiv preprint arXiv:1901.07517*, 2019.

[3] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," *arXiv preprint arXiv:1812.07252*, 2018.

[4] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7559–7566, IEEE, 2018.

[5] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems*, pp. 3981–3989, 2016.

[6] K. Li and J. Malik, "Learning to optimize," *arXiv preprint arXiv:1606.01885*, 2016.

[7] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, pp. 2692–2700, 2015.

[8] L. Gerencsér and H. Hjalmarsson, "Adaptive input design in system identification," in *Proceedings of the 44th IEEE Conference on Decision and Control*, pp. 4988–4993, IEEE, 2005.

[9] L. Gerencsér, H. Hjalmarsson, and J. Mårtensson, "Identification of arx systems with non-stationary inputsasymptotic analysis with application to adaptive input design," *Automatica*, vol. 45, no. 3, pp. 623–633, 2009.

[10] L. Gerencsér, H. Hjalmarsson, and L. Huang, "Adaptive input design for lti systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 5, pp. 2390–2405, 2017.

[11] S. Tu, R. Boczar, and B. Recht, "Minimax lower bounds for h_∞-norm estimation,"

[12] T. J. Mitchell, "An algorithm for the construction of d-optimal experimental designs," *Technometrics*, vol. 16, no. 2, pp. 203–210, 1974.

[13] R. Mehra, "Optimal input signals for parameter estimation in dynamic systems–survey and new results," *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 753–768, 1974.

[14] X. Bombois, M. Gevers, R. Hildebrand, and G. Solari, "Optimal experiment design for open and closed-loop system identification," *Communications in Information and Systems*, vol. 11, no. 3, pp. 197–224, 2011.

[15] A. Czornik, "On discrete-time linear quadratic control," *Systems & control letters*, vol. 36, no. 2, pp. 101–107, 1999.

[16] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, pp. 679–684, 1957.

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

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[19] S. Dean, H. Mania, N. Matni, B. Recht, and S. Tu, "On the sample complexity of the linear quadratic regulator," *arXiv preprint arXiv:1710.01688*, 2017.

[20] Y.-S. Wang, N. Matni, and J. C. Doyle, "A system level approach to controller synthesis," *IEEE Transactions on Automatic Control*, 2019.

[21] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, "RLlib: Abstractions for distributed reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2018.

[22] S. Brakken-Thal, "Gershgorins theorem for estimating eigenvalues," *Source:¡ http://buzzard. ups. edu/courses/2007spring/projects/brakkenthal-paper. pdf*, 2007.

[23] H. Mania, S. Tu, and B. Recht, "Certainty equivalent control of lqr is efficient," *arXiv preprint arXiv:1902.07826*, 2019.