Policy Poisoning in Batch Learning for Linear Quadratic Control Systems via State Manipulation

Courtney M. King, Son Tung Do, and Juntao Chen
Department of Computer and Information Sciences, Fordham University, New York, NY 10023
{cking74,sdo3,jchen504}@fordham.edu

***Abstract*—In this work, we study policy poisoning through state manipulation, also known as sensor spoofing, and focus specifically on the case of an agent forming a control policy through batch learning in a linear-quadratic (LQ) system. In this scenario, an attacker aims to trick the learner into implementing a targeted malicious policy by manipulating the batch data before the agent begins its learning process. An attack model is crafted to carry out the poisoning strategically, with the goal of modifying the batch data as little as possible to avoid detection by the learner. We establish an optimization framework to guide the design of such policy poisoning attacks. The presence of bi-linear constraints in the optimization problem requires the design of a computationally efficient algorithm to obtain a solution. Therefore, we develop an iterative scheme based on the Alternating Direction Method of Multipliers (ADMM) which is able to return solutions that are approximately optimal. Several case studies are used to demonstrate the effectiveness of the algorithm in carrying out the sensor-based attack on the batch-learning agent in LQ control systems.**

***Index Terms*—Policy Poisoning, Sensor Spoofing, Batch Learning, Linear Quadratic System, ADMM**

I. Introduction

Sensors have long been an invaluable tool to automation, forecasting, and efficiency [1], and they continue to grow in number and potential applicability. Our society is highly dependent on sensors to give frequent, consistent, and accurate information. Unfortunately, as AI-based systems become increasingly more popular and complex, so do their vulnerabilities. The sensor is a hot target as such systems rely on large data streams to operate, with sensors performing what is arguably the most important role in the data collection process. The threat of sensor-based attacks is severe [2], and these types of attacks are especially prominent in situations where the sensors are deployed, and the data is sent to different control centers for subsequent processing.

A control system can involve multiple components, and a system that uses learning-based approaches for decision-making, e.g., reinforcement learning, requires large amounts of data, having multiple data streams flowing into the system and between its components. This can leave the system vulnerable if the data becomes corrupted in storage or at any point during its transmission. An attack can effectively be carried out by manipulating the stored data before the agent even begins the learning process or at other stages if any channels within the system are left unprotected. For a typical multi-variable control system, it is even worse if the measurements of these variables (e.g., state and control) can be selectively altered to craft a strategic attack.

To devise an effective learning-based mechanism on a control system, an agent requires information about the current state, action, and cost incurred at each point over a horizon. This data can be collected in batches, with these relevant measurements being taken from its sensors, actuators, and other components, and then used to obtain control policies to automate processes. Such a procedure is highly dependent on the quality of the data. Thus, it is extremely important that the measurements from the system are accurate and that the batch data is not corrupted. If a system is not strongly secured, this presents a weakness that could potentially be exploited by attackers who wish to damage or even inject their own controls into the system's automation. Sensor spoofing is one such case that attempts to manipulate the information collected from the system's sensors [3]. In the case of learning a control policy from batch data, manipulating sensor data can result in control policies that are undesirable or even catastrophic. If an adversary has direct access to the batch data, not only can it significantly alter the intended goal of a learner, but also complete the attack strategically to avoid detection, allowing it to persist on the system [4]. In some cases, an attacker may even use reverse engineering to force a learner to uncover its own malicious policy instead [5].

In this study, we focus on the case of sensor manipulation occurring within a linear quadratic (LQ) system during its batch-learning phase. The learner aims to devise an optimal control policy based on the sampled data, while the attacker aims to trick the learner into learning a malicious policy by poisoning the data samples. With perfect information of the system's learning process, the attacker alters the state values in the batch dataset such that the learner will learn the malicious policy instead of the optimal policy it expects. The attack is carried out strategically, meaning that the adversarial modifications are sparse and the state values are altered as minimally as possible for the attacker to achieve its goal while avoiding detection. This specific case of policy poisoning is formulated as an optimization problem, which poses additional challenges to solve, one of which being bi-linear constraints. We develop an algorithm that makes use of a heuristic method known as Alternating Direction Method of Multipliers (ADMM) to compute the solution and orchestrate the attack. A number of case studies are used to demonstrate the feasibility of the attack model and the effectiveness of the proposed algorithm.

A. Related Works

As the use of AI has become increasingly more common in ordinary applications, so have the threats posed by these techniques [6]. An adversary can carefully devise various attacks such as advanced persistent threats [7], denial of services (DoS) [8], false data injection (FDI) [9], and sensor spoofing [3]. Such attacks can occur in a variety of settings, manipulating critical systems on which we depend on a daily basis, from automated vehicles [10] to medical and other IoT devices [11]. Adversarial attacks on learning-enabled dynamic systems have also become ubiquitous. Understanding these attacks and developing countermeasures to guarantee the safety of the systems are critical [12]. Previous works have explored policy poisoning attacks occurring via manipulation of the system's cost measurements in a discrete-time control system [13], [14]. Our work focuses on the case of policy poisoning through sensor data manipulation on a batch-learning agent in the continuous-time setting, which we believe to be a more realistic and applicable attack scenario.

B. Organization of the Paper

This paper is organized as follows. Section II defines the environment of the problem, introducing the LQ system and batch-learning process. Section III develops the attack model and formulates the policy poisoning scheme as an optimization problem. Section IV presents the algorithm designed to obtain the optimal parameters to craft the attack on the batch learner. Section V shows several case studies, demonstrating the effectiveness of the proposed attack model and implemented algorithm. Section VI concludes the study and gives suggestions for follow-up research.

C. Notations

Let $\mathbb{R}_+$ denote the set of positive real numbers. Let $\mathbb{S}^n$ define the set of symmetric matrices of order n, and further let $\mathbb{S}^n_+$, $\mathbb{S}^n_{++}$ denote those which are also positive semi-definite or positive definite, respectively. The Frobenius norm of a matrix M is denoted by $||M||_F$ (if the subscript is omitted, it denotes the Euclidean norm). The superscript T denotes the transpose of a matrix. I_n stands for the n-dimensional identity matrix. When the context is clear, we use x, u, c instead of $x(t), u(t), c(t)$ to simplify notations. Let M^k denote M to the kth power, and M_k denote its value at the kth iteration. Let $M > 0$ and $M \geq 0$ denote that M is positive definite and positive semi-definite, respectively.

II. Preliminaries

This section defines the problem environment and surrounding context, provides essential background information, and introduces the batch-learning process.

A. LQ Control Basics

Consider a continuous-time linear dynamical system:

$$\begin{aligned} \dot{x} &= Ax + Bu, \\ x(0) &= x_0, \end{aligned} \tag{1}$$

where $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$ are state and control input vectors, respectively, and $A \in \mathbb{R}^{nxn}$ and $B \in \mathbb{R}^{nxm}$ are system matrices.

The cost function to minimize is defined as

$$J(x_0, u) = \int_0^\infty (x^T Q x + u^T R u) dt, \tag{2}$$

where $Q \in \mathbb{S}^n_+$, $R \in \mathbb{S}^m_{++}$ are the pre-defined cost matrices associated with the state and action, respectively. For convenience, we define the instantaneous cost in (2) as

$$c(t) = x(t)^T Q x(t) + u(t)^T R u(t). \tag{3}$$

The optimal value function is defined as:

$$\begin{aligned} V^*(x) :&= \min_u \int_0^\infty (x^T Q x + u^T R u) dt \\ &= x^T P x, \end{aligned} \tag{4}$$

where $P \in \mathbb{S}^n$. We consider the feedback control policy that maps the current state to an action. Specifically, the control input is linear in the state:

$$u = Kx, \tag{5}$$

where $K \in \mathbb{R}^{m \times n}$. We make the following assumption about the system.

Assumption 1. *The system described in* (1) *is stabilizable, i.e., the set*

$$\mathcal{F} = \{K | A + BK \ is \ stable\} \tag{6}$$

is non-empty.

Based on (4), it is well known that the optimal feedback control policy admits the following form:

$$u^* = -R^{-1} B^T P x = Kx, \tag{7}$$

where P is obtained by solving the continuous algebraic Riccati equation (CARE):

$$A^T P + PA - PBR^{-1} B^T P + Q = 0. \tag{8}$$

B. Batch Learning

In batch learning, samples are taken from an existing system to form a dataset of inputs and outputs. In each batch dataset $\mathcal{D}$, we assume that there are N data points:

$$\mathcal{D} = \{(x_k, u_k, c_k) \ for \ k = 0, 1, 2..N-1\}, \tag{9}$$

where

$$x_k \triangleq x(k\Delta t), \ u_k \triangleq u(k\Delta t), \ c_k \triangleq c(k\Delta t).$$

Here, $\Delta t \in \mathbb{R}_+$ is the chosen sampling interval. Each data point is sampled from the system given in (1).

The goal of a learner is to first estimate the unknown system matrices from the batch dataset and then find the optimal feedback control (7). We have the following assumption on the data-generation process.

Assumption 2. *The dataset is generated under a zero-order hold, i.e., $u(t)$, for $t \in [k\Delta t, (k+1)\Delta t)$, $\forall k = 0, 1, .., N-1$, is held constant between each sampling instance.*

Additionally, it is required that the sampling time interval Δt be chosen such that the spectral radius of A is less than $\frac{1}{\Delta t}$. This constraint ensures successful learning of the underlying dynamic system based on data, as detailed in Section III-A.

III. Policy Poisoning by State Manipulation

The learner uses the batch data to form its control policy, K, naturally assuming that the data is untouched and that its procedure will produce a desirable policy in its given environment. The attacker has full knowledge of the learner's processes to compute K and plans to carry out the attack before the learner imports the batch data. The attacker's goal is to trick the learner into learning a malicious policy $K^\dagger$ that deviates from K, which could ultimately guide the system to taking unintended controls, based on (5). Such an attack can be enabled by modifying the state measurements in the batch dataset. More specifically, the attacker can manipulate the control policy learning process by poisoning the benign dataset by replacing the state x with deceptive state $x^\dagger$.

A. Learning the System Dynamics

To devise an optimal attack, the attacker should follow the same procedures as the learner beforehand. As the learner is only given the batch dataset, he has to first perform identification of the underlying system dynamics. Thus, as an intermediate step, the attacker must perform the same system identification on the original dataset. This is achieved through a two-step process that first estimates a discrete-time model from the samples and then determines an equivalent continuous-time model [15], summarized as follows.

Step 1: Learn a discrete-time model from samples.

The given samples are organized into system inputs and outputs assuming the form:

$$x_{k+1} = Fx_k + Gu_k, \tag{10}$$

where

$$F = e^{A\Delta t},\ G = \int_0^{\Delta t} e^{A\tau} d\tau B. \tag{11}$$

The system parameters F and G can be obtained using the least-square estimates:

$$(\hat{F}, \hat{G}) = \underset{F,G}{\operatorname{argmin}} \sum_{k=0}^{N-1} ||(Fx_k + Gu_k) - x_{k+1}||^2. \tag{12}$$

The samples are separated into vectors as follows:

$$X := \begin{bmatrix} x_1^T \\ x_2^T \\ . \\ . \\ x_N^T \end{bmatrix},\ Z := \begin{bmatrix} z_0^T \\ z_1^T \\ . \\ . \\ z_{(N-1)}^T \end{bmatrix}, \tag{13}$$

where X is a vector formed by concatenating the output sequence of the sampled data, and $z_k = [x_k^T, u_k^T]^T$. Under the assumption that $Z^T Z$ is invertible, we can directly obtain the estimations of F and G using the following:

$$[\hat{F}, \hat{G}]^T = (Z^T Z)^{-1} ZX. \tag{14}$$

Algorithm 1 Indirect Method for Continuous Transformation

1: Given initial parameters n_{iter}, $\epsilon = 0.01$
2: Determine $L = F - I$
3: Set $P_0 = I, M_0 = I$, and $i = 0$
4: **repeat**
5: $\quad M_{i+1} = \frac{-iLM_i}{i+1}$
6: $\quad P_{i+1} = P_i + M_{i+1}$
7: $\quad i = i + 1$
8: **until** $i = n_{iter}$ or $||P_{i+1} - P_i||_F \leq \epsilon$
9: Let $A = \frac{PL}{\Delta t}$, $B = \frac{PG}{\Delta t}$
10: **return** A, B

Step 2: Transform the discrete-time model into continuous-time model.

We currently obtain the estimation of F and G. Finding matrices A, B based on (11) requires computing the natural logarithm of a square matrix. This can be done using the indirect method summarized in Algorithm 1.

The cost matrices Q and R may also need to be estimated from the batch data. These can be obtained simply from sampled data x, u, and c as follows[1]:

$$(\hat{Q}, \hat{R}) = \underset{Q,R}{\operatorname{argmin}} \sum_{k=0}^{N-1} ||(x_k^T Q x_k + u_k^T R u_k) - c_k||^2. \tag{15}$$

B. Policy Poisoning

To trick the learner into learning $K^\dagger$, the attacker plans to modify the batch dataset by replacing the original sensor/state measurements, x, with poisoned counterparts, $x^\dagger$. Additionally, the attacker aims to modify the dataset as minimally as possible to avoid detection. It is nontrivial to determine $x^\dagger$ which requires several steps. The attacker first estimates $\hat{A}, \hat{B}$ using the methods described in subsection III-A, then estimates $\hat{Q}, \hat{R}$ from the original batch data by (15). With these learned parameters along with the targeted policy $K^\dagger$, the attacker devises $x^\dagger$ by solving the following optimization problem:

$$\begin{aligned} \min_{x^\dagger, \tilde{A}, P} \quad & \sum_{k=0}^{N-1} ||x_k^\dagger - x_k||^2 \\ \text{s.t.} \quad & \tilde{A}^T P + P(\tilde{A} + \hat{B}K^\dagger) + \hat{Q} = 0, \\ & \hat{R}K^\dagger = -\hat{B}^T P, \\ & P \succeq 0, \\ & \dot{x^\dagger} = \tilde{A}x^\dagger + \hat{B}u, \\ & x^\dagger(0) = x_0. \end{aligned} \tag{16}$$

Based on (16), note that the attacker only changes the measurements of x_k in the batch dataset while leaving the samples of control u_k and cost c_k untouched, for $k = 0, 1, ..., N-1$.

Since c_k depends on x_k, modifying the values of x_k without similarly modifying c_k could result in inconsistencies. This problem can be addressed in two ways. i): It can be assumed that the learner already has perfect knowledge of Q and R and

[1] In our problem setup, only the attacker performs this step. The learner is assumed to hold the true estimates, a priori.

does not estimate Q and R again based on the batch dataset. ii): The cost samples can be subsequently modified according to $c_k^\dagger = x_k^{\dagger T}\hat{Q}x_k^\dagger + u_k^T\hat{R}u_k$. For consistency, the problem (16) should include this modification together with (15) into the constraint. In this work, we assume that the first scenario i) holds to facilitate the analysis.

The problem (16) is challenging to address. The first difficulty is due to the inclusion of the system dynamics in its constraints. The second is the bi-linearity of the first constraint in (16). To address the first challenge, we propose the following two steps.

Step 1: The attacker first finds an optimal $\tilde{A}$ that leads to $K^\dagger$ by solving the following optimization problem:

$$\begin{aligned} \min_{\tilde{A},P} \quad & ||\tilde{A}-\hat{A}||_F^2 \\ \text{s.t.} \quad & \tilde{A}^T P + P(\tilde{A}+\hat{B}K^\dagger) + \hat{Q} = 0, \\ & \hat{R}K^\dagger = -\hat{B}^T P, \\ & P \succeq 0. \end{aligned} \tag{17}$$

Step 2: The attacker then generates the poisoned data according to the following dynamics:

$$\begin{aligned} \dot{x^\dagger} &= \tilde{A}x^\dagger + \hat{B}u, \\ x^\dagger(0) &= x_0. \end{aligned} \tag{18}$$

Note that the poisoned states are sampled based on the same Δt as used in the original batch data generation.

The second challenge left to solve (17) is its non-convexity, due to the bi-linear terms, $\tilde{A}^T P$ and $P\tilde{A}$, in its constraints. Solving this requires us to develop an efficient computational scheme to obtain an approximately optimal solution, which is detailed in the ensuing section.

IV. Solution Method

This section will discuss the method used to find a solution to (17) and obtain the optimal attack parameter given the challenges discussed in Section III-B. This is done by developing an algorithm based on the distributed optimization technique known as ADMM [16].

The algorithm, summarized in Algorithm 2, has three distinct steps. It can be interpreted as one alternatively updating the decision variables as necessary in order to resolve the inseparable terms, $\tilde{A}$ and P.

First, the augmented Lagrangian to (17) is defined as

$$\mathcal{L}_\mu(\tilde{A},P,Z) = ||\tilde{A}-\hat{A}||_F^2 + Tr(Z^T W) + I_P + \frac{\mu}{2}||W||_F^2, \tag{19}$$

with

$$W = \begin{bmatrix} \tilde{A}^T P + P(\tilde{A}+\hat{B}K^\dagger) + \hat{Q} \\ \hat{R}K^\dagger + \hat{B}^T P \end{bmatrix}, \tag{20}$$

and

$$I_P = \begin{cases} 0 & \text{if} \quad P \succeq 0 \\ +\infty & \text{o.w.} \end{cases}, \tag{21}$$

where Z is a dual variable with an appropriate dimension matching W, and $\mu > 0$ is a penalty parameter. Matrix W captures the constraints in (17) and I_P is the step function.

Algorithm 2 ADMM-based Approach

1: Given initial parameters: $\tilde{A}_0$, P_0, Z_0 penalty parameter μ, number of iterations n_{iter}
2: **for** $i = 0, 1, ...n_{iter-1}$ **do**
3: Let $\tilde{A}_{i+1} = \arg\min_{\tilde{A}} \mathcal{L}_\mu(\tilde{A}, P_i, Z_i)$
4: Let $P_{i+1} = \arg\min_P \mathcal{L}_\mu(\tilde{A}_{i+1}, P, Z_i)$
5: Let $Z_{i+1} =$
$$Z_i + \mu \begin{bmatrix} \tilde{A}_{i+1}^T P_{i+1} + P_{i+1}(\tilde{A}_{i+1} + \hat{B}K^\dagger) + \hat{Q} \\ \hat{R}K^\dagger + \hat{B}^T P_{i+1} \end{bmatrix}$$
6: **end for**
7: **return** $\tilde{A}_{n_{iter}}$

The three major steps in Algorithm 2 are presented below.

1) $\tilde{A}$ Step: The first update step can be expressed as solutions to the following convex optimization problem:

$$\tilde{A}_{i+1} = \arg\min_{\tilde{A}} \ \mathcal{L}_\mu(\tilde{A}, P_i, Z_i), \tag{22}$$

and more specifically:

$$\begin{aligned} \min_{\tilde{A}} \ & ||\tilde{A}-\hat{A}||_F^2 + \\ & \frac{\mu}{2}\left\| \begin{bmatrix} \tilde{A}^T P_i + P_i(\tilde{A}+\hat{B}K^\dagger) + \hat{Q} + \frac{1}{\mu}Z_i^1 \\ \hat{R}K^\dagger + \hat{B}^T P_i + \frac{1}{\mu}Z_i^2 \end{bmatrix} \right\|_F^2, \end{aligned} \tag{23}$$

with $Z_i = (Z_i^1, Z_i^2)$. Notice that the trace term in the original Lagrangian is absorbed into the Frobenius norm term due to their equivalence. This step generates an $\tilde{A}$ that is close to $\hat{A}$ while leading to the desired policy $K^\dagger$.

2) P Step: The second step is to solve the following convex optimization problem:

$$P_{i+1} = \arg\min_P \ \mathcal{L}_\mu(\tilde{A}_{i+1}, P, Z_i), \tag{24}$$

and more specifically:

$$\begin{aligned} \min_P \ & \left\| \begin{bmatrix} \tilde{A}_{i+1}^T P + P(\tilde{A}_{i+1} + \hat{B}K^\dagger) + \hat{Q} + \frac{1}{\mu}Z_i^1 \\ \hat{R}K^\dagger + \hat{B}^T P + \frac{1}{\mu}Z_i^2 \end{bmatrix} \right\|_F^2, \\ \text{s.t.} \ & P \succeq 0. \end{aligned} \tag{25}$$

This step results in P_{i+1} which is an approximate solution to the Riccati equation, yielding the attacker's desired policy $K^\dagger$.

3) Z Step: This step performs the dual update as follows:

$$\begin{aligned} Z_{i+1} = Z_i + \\ \mu \begin{bmatrix} \tilde{A}_{i+1}^T P_{i+1} + P_{i+1}(\tilde{A}_{i+1} + \hat{B}K^\dagger) + \hat{Q} + \frac{1}{\mu}Z_i^1 \\ \hat{R}K^\dagger + \hat{B}^T P_{i+1} + \frac{1}{\mu}Z_i^2 \end{bmatrix}. \end{aligned} \tag{26}$$

The algorithm is guaranteed to converge to a stable point given that penalty parameter μ is sufficiently large [17]. Next, we will use several case studies to showcase the effectiveness of Algorithm 2 for policy poisoning in batch learning.

V. Case studies

In this section, we use several case studies to illustrate the effectiveness of the developed policy poisoning scheme described in Section III-B using the algorithm developed in Section IV. We leverage CVXPY [18] to solve the corresponding optimization problems when using Algorithm 2.

A. Case 1- Random Case

We first demonstrate the attack on an LQ system with random parameters. The system starts at the initial state $x_0 = [0.5, -0.5, 0.5, -0.5]^T$. The sampling interval is $\Delta t = 0.01$, $Q = I_4$, $R = 0.5I_2$, and the system matrices are:

$$A = \begin{bmatrix} 0.59 & 0.13 & 0.33 & 0.76 \\ 0.63 & -0.33 & 0.32 & -0.05 \\ -0.03 & 0.14 & 0.05 & 0.49 \\ 0.26 & 0.04 & 0.15 & 0.11 \end{bmatrix}, B = \begin{bmatrix} 1.49 & -0.21 \\ 0.31 & -0.85 \\ -2.55 & 0.65 \\ 0.86 & -0.74 \end{bmatrix}.$$

The optimal control policy can be obtained as:

$$K^* = \begin{bmatrix} -2.87 & -0.38 & -0.34 & -1.24 \\ 4.32 & 1.14 & 3.63 & 4.71 \end{bmatrix}.$$

The learned system matrix A based on the uncompromised dataset is:

$$\hat{A} = \begin{bmatrix} 0.60 & 0.13 & 0.33 & 0.76 \\ 0.63 & -0.33 & 0.32 & -0.05 \\ -0.03 & 0.14 & 0.05 & 0.49 \\ 0.26 & 0.04 & 0.15 & 0.11 \end{bmatrix}.$$

With imperfect information on the system dynamics, the learner's expected control policy without attack is

$$\hat{K} = \begin{bmatrix} -2.86 & -0.38 & -0.34 & -1.24 \\ 4.29 & 1.13 & 3.62 & 4.69 \end{bmatrix},$$

which closely resembles the optimal policy K^*.

Assume that the attacker's target policy is

$$K^\dagger = \begin{bmatrix} -0.11 & 0.99 & 0.5 & -5.55 \\ 0.53 & 0.26 & 2.07 & 9.36 \end{bmatrix}.$$

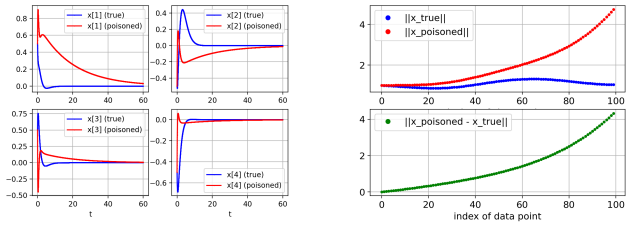
The attacker then uses the steps in Section III-B to generate the poisoned dataset. Based on Algorithm 2, it yields

$$\tilde{A} = \begin{bmatrix} 0.17 & 0.05 & 0.02 & 0.41 \\ 0.41 & 0.24 & -0.29 & -0.35 \\ -0.43 & 0.05 & 0.29 & 0.69 \\ 0.57 & 0.46 & 0.16 & 2.57 \end{bmatrix}.$$

The learner unknowingly learns from the poisoned dataset and reaches the policy:

$$\hat{K} = \begin{bmatrix} -0.11 & 0.99 & 0.5 & -5.57 \\ 0.53 & 0.26 & 2.06 & 9.22 \end{bmatrix}.$$

Thus, the attacker successfully deceives the learner into learning the targeted policy. As shown in Fig. 1, the attack causes a significant change in the system's state trajectories. In addition, the poisoned system takes much longer to stabilize.



(a) States (b) Attack cost

Fig. 1. Poisoning a random system

B. Case 2- Active Suspension System

Next, we attempt to poison the active suspension system on a vehicle. The active suspension system consists of a car body with mass m_b and wheel assembly with mass m_w, connected by spring k_s and damper b_s. Spring k_t models the compressibility of the pneumatic tire. x_b and x_w are body travel and wheel travel, respectively. The force f_s (in kilo Newtons) between the body and the wheel is the variable that can be controlled. Denoting $(x_1, x_2, x_3, x_4) := (x_b, \dot{x}_b, x_w, \dot{x}_w)$, we get the linearized state-space representation [19]:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_1 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{k_s}{m_b} & -\frac{b_s}{m_b} & \frac{k_s}{m_b} & \frac{b_s}{m_b} \\ 0 & 0 & 0 & 1 \\ \frac{k_s}{m_w} & \frac{b_s}{m_w} & \frac{-k_s - k_t}{m_w} & -\frac{b_s}{m_w} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{10^3}{m_b} \\ 0 \\ \frac{-10^3}{m_w} \end{bmatrix} \begin{bmatrix} f_s \end{bmatrix}, \tag{27}$$

where the control is $u = f_s$. With $m_b = 300\ kg$, $m_w = 60\ kg$, $b_s = 1000\ N/m/s$, $k_s = 16000\ N/m$, $k_t = 190000\ N/m$, the corresponding A matrix is:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -53.33 & -3.33 & 53.33 & 3.33 \\ 0 & 0 & 0 & 1 \\ 266.67 & 16.67 & -3433.33 & -16.67 \end{bmatrix}.$$

In addition, the initial state is $x_0 = [1, -10, 0.3, 10]^T$; the sampling interval is $\Delta t = 0.00005$; and the cost matrices are $Q = I_4, R = 0.1I_1$. To this end, the optimal control policy is

$$K^* = \begin{bmatrix} -0.31 & -2.57 & 30.6 & 2.22 \end{bmatrix}.$$

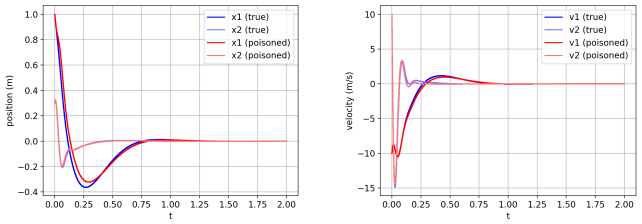
The estimated system matrix A in batch learning admits

$$\hat{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -53.31 & -3.33 & 53.04 & 3.33 \\ 0 & 0 & 0 & 1 \\ 266.53 & 16.66 & -3431.87 & -16.74 \end{bmatrix}.$$

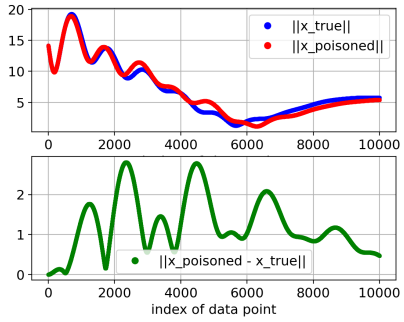
With imperfect information on the system dynamics, the learner's expected control policy is

$$\hat{K} = \begin{bmatrix} -0.31 & -2.57 & 30.52 & 2.21 \end{bmatrix}.$$

The attacker aims to manipulate the learner into learning the parameter A corresponding to different values of spring and



(a) Position (b) Velocity



(c) Attack cost

Fig. 2. Poisoning an active suspension system

damper coefficients. The attacker chooses $k_s = 2000\ N/m$, which yields

$$K^\dagger = \begin{bmatrix} -1.74 & -2.53 & 32.1 & 2.26 \end{bmatrix}.$$

The attacker then uses the steps in Section III-B to generate the poisoned dataset. Based on Algorithm 2, it yields

$$\tilde{A} = \begin{bmatrix} 0 & 1.01 & 0 & 0 \\ -53.33 & -3.49 & 53.02 & 3.46 \\ 0 & 0 & 0 & 1.06 \\ 266.53 & 16.68 & -3431.87 & -16.73 \end{bmatrix}.$$

The learner ends up with the poisoned control policy:

$$\hat{K} = \begin{bmatrix} -1.7 & -2.62 & 30.8 & 2.25 \end{bmatrix}.$$

Hence, the learner is successfully guided to the attacker's desired policy with a small amount of errors. From Fig. 2, the poisoned system takes slightly longer to reach equilibrium. For a suspension system, this decreases the efficiency of the system and results in less comfort for the driver.

VI. Conclusion

In this study, we have pointed out a potential vulnerability of a batch-learning agent within an LQ system and developed a strategic attack model to effectively perform policy poisoning of the system. A computationally efficient algorithm has been designed to compute the optimal parameters to carry out the proposed attack on the system during its batch learning phase. Several case studies have demonstrated that the proposed policy poisoning scheme was successful in forcing the learner to learn a target policy desired by the attacker, significantly altering the system's performance. It is therefore crucial for an LQ system to actively protect its stored data, and specifically its sensor data, for trustworthy batch learning. Future work would include exploring methods of detection or protection from the proposed attack model and extending the poisoning scheme to online learning-based systems.

References

[1] J. Borenstein, H. R. Everett, L. Feng, and D. Wehe, "Mobile robot positioning: Sensors and techniques," *Journal of Robotic Systems*, vol. 14, no. 4, pp. 231–249, 1997.

[2] D. Davidson, H. Wu, R. Jellinek, V. Singh, and T. Ristenpart, "Controlling UAVs with sensor input spoofing attacks," in *10th USENIX Workshop on Offensive Technologies*, 2016.

[3] T. Zhang and Q. Zhu, "Strategic defense against deceptive civilian gps spoofing of unmanned aerial vehicles," in *International Conference on Decision and Game Theory for Security*. Springer, 2017, pp. 213–233.

[4] Y.-C. Liu, G. Bianchin, and F. Pasqualetti, "Secure trajectory planning against undetectable spoofing attacks," *Automatica*, vol. 112, p. 108655, 2020.

[5] A. Rakhsha, G. Radanovic, R. Devidze, X. Zhu, and A. Singla, "Policy teaching via environment poisoning: Training-time adversarial attacks against reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 7974–7984.

[6] H. Liu, Y. Wang, W. Fan, X. Liu, Y. Li, S. Jain, Y. Liu, A. Jain, and J. Tang, "Trustworthy AI: A computational perspective," *ACM Transactions on Intelligent Systems and Technology*, vol. 14, no. 1, pp. 1–59, 2022.

[7] J. Chen and Q. Zhu, "Security as a service for cloud-enabled internet of controlled things under advanced persistent threats: a contract design approach," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 11, pp. 2736–2750, 2017.

[8] Y. Yuan, F. Sun, and Q. Zhu, "Resilient control in the presence of dos attack: Switched system approach," *International Journal of Control, Automation and Systems*, vol. 13, no. 6, pp. 1423–1435, 2015.

[9] H. Lin, J. Lam, and Z. Wang, "Secure state estimation for systems under mixed cyber-attacks: Security and performance analysis," *Information Sciences*, vol. 546, pp. 943–960, 2021.

[10] J. Chen and Q. Zhu, "Control of multilayer mobile autonomous systems in adversarial environments: A games-in-games approach," *IEEE Transactions on Control of Network Systems*, vol. 7, no. 3, pp. 1056–1068, 2019.

[11] J. Pawlick and Q. Zhu, "Strategic trust in cloud-enabled cyber-physical systems with an application to glucose control," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 12, pp. 2906–2919, 2017.

[12] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, pp. 411–444, 2022.

[13] Y. Huang and Q. Zhu, "Reinforcement learning for linear quadratic control is vulnerable under cost manipulation," *arXiv preprint arXiv:2203.05774*, 2022.

[14] Y. Ma, X. Zhang, W. Sun, and J. Zhu, "Policy poisoning in batch reinforcement learning and control," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[15] N. K. Sinha, "Identification of continuous-time systems from samples of input-output data: An introduction," *Sadhana*, vol. 25, no. 2, pp. 75–83, 2000.

[16] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[17] W. Gao, D. Goldfarb, and F. E. Curtis, "ADMM for multiaffine constrained optimization," *Optimization Methods and Software*, vol. 35, no. 2, pp. 257–303, 2020.

[18] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[19] Mathworks, "Robust control of active suspension," https://www.mathworks.com/help/robust/gs/active-suspension-control-design.html, 2022, accessed: 2022-10-26.