Adversarial Manipulation of Learning in Linear-Quadratic Zero-Sum Differential Games via Cost Poisoning

Son Tung Do, Gabrielle Ebbrecht, and Juntao Chen
Department of Computer and Information Sciences, Fordham University, New York, NY 10023
E-mail: {sdo3, gebbrecht, jchen504}@fordham.edu

***Abstract*—It is important to study attacks on games to gain insights into control system vulnerabilities and better understand how adversaries may optimize their malicious behavior to evade detection. Learning games are susceptible to a number of attacks, including deceptive attacks that misguide an algorithm to learn from inaccurate data. This work investigates policy poisoning attacks in a linear-quadratic (LQ) zero-sum differential game in which two players learn their control policies from batch data. Acting as an adversary, we design malicious policies to manipulate cost measurements in the data before players begin the learning process with the aim of demonstrating vulnerabilities to cost-poisoning attacks and their effects on compromised systems. The poisoning strategy is formulated as an optimization problem that includes a constraint on deviations from original batch data as a means to avoid detection. We leverage case studies, including a pursuit-evasion game, to further evidence attack feasibility and impacts.**

I. Introduction

In the game of chess, one of the most well-known examples of a zero-sum game, the sum of the payoffs for all players is constant, and any one player's gain directly comes at the expense of another player. There is a clear winning condition, a checkmate, where one's victory seals the other's loss. A differential zero-sum game would further involve continuous strategic decision-making and dynamic interaction where player strategies directly influence the system, in contrast to the concrete steps and finite nature of chess. In cybersecurity contexts, differential games can model antagonistic dynamics to facilitate strategic planning. Linear-quadratic games, a subclass of differential games, are commonly used to model resource management and economic control in dynamic markets, and solve optimal control problems in the field of engineering. These dynamic decision-making problems are distinguished as having linear dynamics and quadratic cost functions.

For a control system to work well in dynamic environments, it is essential to establish a preferred path and regularly adjust system inputs in order to follow the predefined trajectory. In games, this allows players to adapt to changes in state. For example, in the context of cruise control, the control system must account for changing road conditions in adhering to a vehicle's desired speed. To do this, the control system adjusts its inputs, or its acceleration, based on environmental factors like inclines and slopes. This can be achieved via implementation of a linear-quadratic regulator (LQR) controller and application of filters to yield accurate state estimates [1], [2]. In cases like this, it is important to consider the cost of applying the control input as the state varies over time. The control applied should also adhere to objectives like resource conservation, or a defined schedule, which can be accomplished by learning a control policy that optimizes cost. When little is known about the given environment, batch data from an existing system is useful for estimating system and cost matrices.

Policy learning in observed games, such as the famous two-player zero-sum game, is an interesting application of this scenario that is widely used to model economic situations and military conflict. In highly competitive environments like these, the cost is a crucial element in the learning and decision-making process. Though an LQ control system may show some robustness to unexpected inaccuracies in cost parameters, the mathematical design of its control algorithm may cause the system to make suboptimal decisions if the accuracy of its input is compromised. If the data collected is corrupted in storage or transmission, the players may poorly estimate the system and cost parameters, leading to less effective control policies which may influence game outcome overall, especially when data is intentionally manipulated by an adversary that injects its own objectives into the game prior to batch learning.

Manipulation of the cost data can be leveraged by one player to increase the other player's loss, thus increasing their own gain. These attacks have historically been used in stealth, diffusion, and defense tactics in the context of warfare [3], and can also be observed in nature [4]. As digital control systems become more prevalent for tasks spanning industrial automation, transportation, and healthcare, cyberattacks may threaten system integrity by disrupting data transmission or manipulating the data collected from sensors, and pose a risk to physical safety [5], [6]. Hence, it is important to study these types of attacks as they are highly plausible in various contexts and may have profound effects.

This study focuses on cost manipulation which occurs within a linear-quadratic (LQ) zero-sum game during its batch learning phase. Players aim to devise optimal strategies, or feedback-control policies, based on observations of an existing game. Simultaneously, an adversary attempts to intercept this process and trick the players into learning malicious strategy sets by poisoning the data available to the players. By strategically manipulating the cost measurements, we show how an adversary is able to reach this objective. Two case studies, including the well-known pursuit-evasion game, demonstrate the effectiveness of the attack model with attack results.

A. Related Works

When a control system learns from batch data, an adversary is able to perform policy poisoning by manipulating any of the available variables, including factors such as sensor readings, actuator inputs, or cost incurred. Related works have explored several variations of policy poisoning, including state manipulation in a continuous-time system [7] and cost manipulation in a discrete-time system [8], in a single-controller context. These concepts can be applied to the study of linear-quadratic differential games with multiple players [9], [10], which allows for considering additional system impacts and complexity. Different attack intents may also be considered in this context, such as the case when one player aims to undermine the other to gain an advantage. Pursuit-evasion games [11] are a widely-studied and well-known example of the two-player game case, which this work will leverage to demonstrate the vulnerability of two-player zero-sum games in continuous time to policy poisoning by means of cost manipulation. We consider this scenario to be applicable and therefore worthy of study.

B. Organization of the Paper

This paper is organized as follows. Section II defines the environment of the problem, introducing the LQ zero-sum differential game and batch learning process. Section III develops the attack model and formulates the policy poisoning scheme as an optimization problem. Section IV consists of two case studies, demonstrating the effectiveness of the proposed attack model. Section V concludes the study and gives suggestions for compelling follow-up research.

C. Notations

Let $\mathbb{R}$ denote the set of all real numbers, with $\mathbb{R}_+$, $\mathbb{R}_{++}$ further denoting those that are non-negative or positive, respectively. 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. Also, let an n-dimensional identity matrix be denoted by I_n. For some matrix M, let $M \succeq 0$ and $M \succ 0$ denote semi-positive and positive definiteness, respectively. The Frobenius norm of a matrix M is denoted by $||M||_F$. The superscript T denotes the transpose of a matrix.

II. Preliminaries

We first introduce the foundations of LQ zero-sum differential games and discuss the batch learning process used by players to form their strategies.

A. LQ Zero-Sum Game Fundamentals

An LQ zero-sum differential game is described over the following continuous-time dynamical system:

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

with the state as $x \in \mathbb{R}^n$, the input of player 1 as $u \in \mathbb{R}^m$, and the input of player 2 as $d \in \mathbb{R}^p$, and $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, and $D \in \mathbb{R}^{n\times p}$ are the corresponding system matrices.

The game evolves according to the following cost function:

$$J(x_0,u,d) = \int_0^\infty (x^T Qx + u^T Ru - d^T Md)dt, \tag{2}$$

where $Q \in \mathbb{S}^n_{++}, R \in \mathbb{S}^m_+, M \in \mathbb{S}^p_+$.

The instantaneous cost is hence denoted by:

$$c = x^T Qx + u^T Ru - d^T Md \tag{3}$$

We aim to learn the feedback control policies of the players that map the current state to action. Specifically, the decisions of the players are linear in the state:

$$\begin{aligned} u &= K^u x, \\ d &= K^d x, \end{aligned} \tag{4}$$

where $K^u \in \mathbb{R}^{m\times n}$ and $K^d \in \mathbb{R}^{p\times n}$.

Assumption 1. *The set of admissible control policies is stabilizable, i.e., the set*

$$\mathscr{F} = \{(K^u, K^d) | A + BK^u + DK^d \text{ is stable}\} \tag{5}$$

is non-empty.

In a zero-sum differential game, the first player chooses actions which minimize the game's cost function, while the second player seeks to maximize it. Thus, the desired control policies satisfy:

$$\min_u \max_d \; J(x_0,u,d). \tag{6}$$

A Nash equilibrium (NE) exists when neither player can benefit from changing their strategy, given their opponent's strategy. As the game's NE indicates optimal play for each player, it can also be said that the game is stabilizing. It is reasonable to assume the players will continue the game using this set of strategies, as any strategic change would yield a less favorable result.

NE is a critical concept to the study of LQ zero-sum games as it facilitates the analysis of player performance and interaction. By studying attacks in the context of a game's NE, the ways in which player's optimal strategies are manipulated can be observed, informing future approaches to defend game integrity. To this end, we present the definition of NE below.

Definition 1 (Feedback Nash Equilibrium)**.** *A strategy pair (u^*, d^*) is called a feedback NE in the considered LQ zero-sum differential game if*

$$J(x_0,u^*,d) \le J(x_0,u^*,d^*) \le J(x_0,u,d^*), \tag{7}$$

which is equivalent to $J(x_0,u^*,d^*) = \min_u \max_d \; J(x_0,u,d)$.

The game's optimal value function can be described by:

$$\begin{aligned} V^*(x) :&= \min_u \max_d \int_0^\infty (x^T Qx + u^T Ru - d^T Md)dt \\ &= x^T Px, \end{aligned} \tag{8}$$

where P is a symmetric matrix.

Based on (8), it is well known that the optimal strategies of the players admit the following solutions [12, Chapter 6]:

$$\begin{aligned} u^* &= -R^{-1}B^T Px := K^u x, \\ d^* &= M^{-1}D^T Px := K^d x, \end{aligned} \tag{9}$$

where P is obtained by solving the following Generalized Algebraic Riccati Equation (GARE):

$$A^T P + PA - P(BR^{-1}B^T - DM^{-1}D^T)P + Q = 0. \tag{10}$$

Under Assumption 1, P admits a unique symmetric solution to (10), and the resulting NE in (9) leads to stabilized state trajectories [13].

B. Batch Learning

Let us consider a two-player game that is sampled for training. Assuming there are N data points available, constituting a dataset $\mathscr{D}$:

$$\mathscr{D} = \{(x_k, u_k, d_k, c_k)...k = 1, 2, ..., N\}. \tag{11}$$

where

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

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 the players is to first estimate the system matrices, A, B, D and Q, R, M from the batch data, then find their optimal feedback controllers, K^u, K^d based on (9) by solving the Riccati equation (10). If the data is incorrect, the estimated system matrices will also be inaccurate, adversely affecting the feedback controllers and player decision-making.

III. Policy Poisoning by Cost Manipulation

In this section, we introduce the batch learning and policy learning processes used by the players and discuss the attack model used to achieve the proposed policy poisoning.

The players use batch data to form their control policies, K^u and K^d, assuming that the data is untouched and that their procedures will produce desirable policies for the given environment. An attacker has full knowledge of the players' processes for computing K^u and K^d and plans to carry out an attack before they import the batch data. The attacker's goal is to trick the players into learning the malicious policies $K^{u\dagger}$ and $K^{d\dagger}$ that deviate from K^u and K^d, which may guide the players in taking unintended controls as based on (4). Such an attack can be conducted by poisoning the batch dataset by replacing benign cost measurements with the attacker's deceptive cost $c^\dagger$.

A. Learning the Game Dynamics

To devise an optimal attack, the attacker should follow the same procedures as the players beforehand. As the players are only given the batch dataset, they must 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 [14], 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:

$$x_{k+1} = Fx_k + Gu_k + Hd_k, \tag{12}$$

where

$$F = e^{A\Delta t},\ G = \int_0^{\Delta t} e^{A\tau} d\tau B,\ H = \int_0^{\Delta t} e^{A\tau} d\tau D. \tag{13}$$

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

$$(\hat{F}, \hat{G}, \hat{H}) = \underset{F,G,H}{\operatorname{argmin}} \sum_{k=0}^{N-1} ||(Fx_k + Gu_k + Hd_k) - x_{k+1}||^2. \tag{14}$$

The samples are separated into vectors as follows:

$$X := \begin{bmatrix} x_1^T \\ x_2^T \\ . \\ . \\ x_N^T \end{bmatrix},\ Z := \begin{bmatrix} z_0^T \\ z_1^T \\ . \\ . \\ z_{(N-1)}^T \end{bmatrix}, \tag{15}$$

where X is a vector formed by concatenating the output sequence of the sampled data, and $z_k = [x_k^T, u_k^T, d_k^T]^T$.

Under the assumption that $Z^T Z$ is invertible, we can directly obtain the estimations of F, G and H using the following:

$$[\hat{F}, \hat{G}, \hat{H}]^T = (Z^T Z)^{-1} ZX. \tag{16}$$

Step 2: Transform the discrete-time model into a continuous-time model. We currently retain the estimation of F, G, H. Finding matrices A, B, D based on step 1 requires computing the natural logarithm of a square matrix using the indirect method summarized in Algorithm 1.

Algorithm 1 Computing System Matrices

```
Given initial parameters n_iter, ε = 0.01
Determine L = F − I
Set P_0 = I, M_0 = I and i = 0
repeat
    M_{i+1} = −iLM_i / (i+1)
    P_{i+1} = P_i + M_{i+1}
    i = i + 1
until i = n_iter or ||P_{i+1} − P_i||_F ≤ ε
Let A = PL/Δt, B = PG/Δt, D = PH/Δt
return A, B, D
```

Next, the cost matrices Q, R, and M need to be estimated from the batch data. These can be obtained simply from sampled data x, u, d, and c as follows:

$$(\hat{Q}, \hat{R}, \hat{M}) = \underset{Q,R,M}{\operatorname{argmin}} \sum_{k=0}^{N-1} ||(x_k^T Q x_k + u_k^T R u_k - d_k^T M d_k) - c_k||^2. \tag{17}$$

B. Policy Poisoning

To manipulate c_k accordingly, the attacker considers the cost matrices Q,R,M. First, the attacker estimates $\hat{A},\hat{B},\hat{D}$ and $\hat{Q},\hat{R},\hat{M}$ using the methods shown in (12) and (17), respectively. Based on these learned parameters and the attacker's target policies, $K^{u\dagger},K^{d\dagger}$, the data poisoning strategy can be summarized by the following optimization problem:

$$\begin{aligned}\min_{c^\dagger,P,\tilde{Q},\tilde{R},\tilde{M}} \quad & \sum_{k=0}^{N-1} ||c_k^\dagger - c_k||^2 \\ \text{s.t.} \quad & \hat{A}^T P + P(\hat{A}+\hat{B}K^{u\dagger}+\hat{D}K^{d\dagger}) + \tilde{Q} = 0, \\ & \tilde{R}K^{u\dagger} = -\hat{B}^T P, \\ & \tilde{M}K^{d\dagger} = \hat{D}^T P, \\ & P \succeq 0, \tilde{Q} \succeq 0, \tilde{R} \succeq I_p, \tilde{M} \succeq I_m, \\ & c_k^\dagger = x_k^T \tilde{Q} x_k + u_k^T \tilde{R} u_k - d_k^T \tilde{M} d_k, \forall k = 1,2,...,N. \end{aligned} \tag{18}$$

This can be understood as the attacker finding the optimal $c_k^\dagger$ to replace the observed c_k for each data point to reach the desired objective without detection, addressing the trade-off between maliciousness and stealth. The first constraint implants the attacker's target policies into the Riccati equation, and the second and third describe the optimal strategies for each player. The fourth constraint ensures that the cost parameters meet the positive and semi-definiteness requirements to solve the problem. The last constraint defines the projected value of the poisoned cost elements across the batch data.

We have the following result regarding the formulated attacker's problem.

Proposition 1. *The problem shown in* (18) *is convex.*

Proof. It is observed that the quadratic objective function is convex. Next, it will be demonstrated that the problem constraints form a convex set. Suppose (P_1,Q_1,R_1,M_1) and (P_2,Q_2,R_2,M_2) satisfy the problem's constraints. We need to show that for any $0 \le \beta \le 1$, $(\tilde{P},\tilde{Q},\tilde{R},\tilde{M})$ also satisfies the constraints, where $\tilde{P} = \beta P_1 + (1-\beta)P_2$, $\tilde{Q} = \beta Q_1 + (1-\beta)Q_2$, $\tilde{R} = \beta R_1 + (1-\beta)R_2$, and $\tilde{M} = \beta M_1 + (1-\beta)M_2$. We know that

$$\begin{aligned} Q_1 &= -\hat{A}^T P_1 - P_1(\hat{A}+\hat{B}K^{u\dagger}+\hat{D}K^{d\dagger}), \\ Q_2 &= -\hat{A}^T P_2 - P_2(\hat{A}+\hat{B}K^{u\dagger}+\hat{D}K^{d\dagger}). \end{aligned}$$

Multiplying both sides of the first equality by β and both sides of the second by $(1-\beta)$ yields

$$\begin{aligned} &\beta Q_1 + (1-\beta)Q_2 = -\hat{A}^T[\beta P_1 + \\ &(1-\beta)P_2] - ([\beta P_1 + (1-\beta)P_2](\hat{A}+\hat{B}K^{u\dagger}+\hat{D}K^{d\dagger})), \end{aligned}$$

which leads to

$$\tilde{Q} = -\hat{A}^T \tilde{P} - \tilde{P}(\hat{A}+\hat{B}K^{u\dagger}+\hat{D}K^{d\dagger}).$$

For the next constraint, we can follow the same procedure. We know that

$$\begin{aligned} R_1 K^{u\dagger} &= -\hat{B}^T P_1, \\ R_2 K^{u\dagger} &= -\hat{B}^T P_2. \end{aligned}$$

Multiplying both sides of the first equality by β and both sides of the second by $(1-\beta)$ yields $[\beta R_1 + (1-\beta)R_2]K^{u\dagger} = -\hat{B}^T[\beta P_1 + (1-\beta)P_2]$, which is equivalent to

$$\tilde{R}K^{u\dagger} = -\hat{B}^T \tilde{P}.$$

The proof for the remaining constraints follows similarly, and thus the constraints of (18) form a convex set. □

Directly addressing problem (18) could be computationally prohibitive when the size of the batch dataset is large. This motivates us to develop an alternative two-step approach to tackle the optimization problem (18), as detailed below.

Step 1: the attacker first solves the following problem to find $P,\tilde{Q},\tilde{R},\tilde{M}$ that will lead to $K^{u\dagger}$, $K^{d\dagger}$:

$$\begin{aligned}\min_{P,\tilde{Q},\tilde{R},\tilde{M}} \quad & ||\tilde{Q}-\hat{Q}||_F^2 + ||\tilde{R}-\hat{R}||_F^2 + ||\tilde{M}-\hat{M}||_F^2 \\ \text{s.t.} \quad & \hat{A}^T P + P(\hat{A}+\hat{B}K^{u\dagger}+\hat{D}K^{d\dagger}) + \tilde{Q} = 0, \\ & \tilde{R}K^{u\dagger} = -\hat{B}^T P, \\ & \tilde{M}K^{d\dagger} = \hat{D}^T P, \\ & P \succeq 0, \tilde{Q} \succeq 0, \tilde{R} \succeq I_p, \tilde{M} \succeq I_m. \end{aligned} \tag{19}$$

Note that the cost function in (19) captures the minimal deviation of the players' cost matrices that yields the attacker's desired control policy. This in turn preserves the deceptive behavior of the attacker in compromising the cost data.

Step 2: The attacker then uses the parameters learned in step 1 to generate the poisoned dataset:

$$c_k^\dagger = x_k^T \tilde{Q} x_k + u_k^T \tilde{R} u_k - d_k^T \tilde{M} d_k, \tag{20}$$

based on the same sampling interval Δt used in generating the original batch dataset.

Next, we use case studies to demonstrate the effects of the proposed policy poisoning scheme on two players learning their control policies, comparing outcomes between clean and poisoned batch data used in the learning.

IV. Case Studies

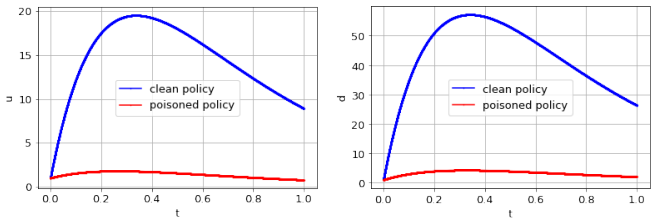
In this section, case studies show the effectiveness of the developed policy poisoning scheme described in Section III-B. The attack can be applied to a wide range of problems, and we present two examples. The first case study leverages a general model for simplicity, and the second demonstrates the more tenable example of a pursuit-evasion game. We leverage CVXPY [15] to solve each optimization problem.

A. Small Case

In this case study, we demonstrate the attack method with the system dynamics initialized as follows. Let the initial state of the system be defined as $x_0 = [1,1]^T$, the sampling interval as $\Delta t = 2.5e-5$, and the matrices governing the system dynamics as:

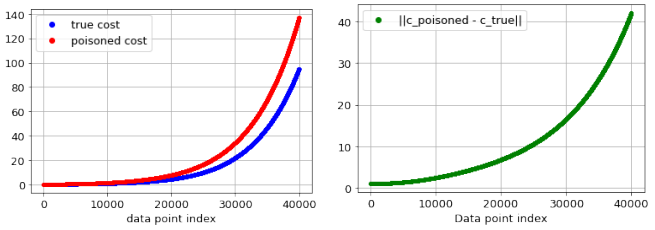
$$A = \begin{bmatrix} 3 & -2 \\ 2 & -4 \end{bmatrix},\ B = \begin{bmatrix} 1 \\ 2 \end{bmatrix},\ D = \begin{bmatrix} 1 & 0 \\ 2 & -2 \end{bmatrix}.$$

The associated cost matrices in the objective are selected as: $Q = I_2,\ R = I_1,\ M = 5I_2$. The optimal policies K^{u*},K^{d*} used



(a) Player 1 trajectory (b) Player 2 trajectory

Fig. 1. The differences of the game play of each player shown in trajectories produced from the policies learned from the clean and poisoned dataset.



(a) Cost values (b) Attack cost

Fig. 2. (a): The cost observations in the original batch dataset and poisoned dataset, in blue and red respectively, at each data point index. (b): The attack cost defined as the difference between the two cost observations, in green.

to generate the batch data for the learner are found by solving the GARE:

$$K^{u*} = \begin{bmatrix} 361.25 \\ -121.69 \end{bmatrix}, \; K^{d*} = \begin{bmatrix} -72.25 & 24.34 \\ -148.01 & 49.67 \end{bmatrix}.$$

After estimating the system and cost parameters from the untouched dataset, the players expect the following policies:

$$\hat{K}^{u} = \begin{bmatrix} 286.05 \\ -96.40 \end{bmatrix}, \; \hat{K}^{d} = \begin{bmatrix} -57.54 & 19.39 \\ -117.76 & 39.49 \end{bmatrix}.$$

The adversary has the following target policies it wishes to inject into the batch data:

$$K^{u\dagger} = \begin{bmatrix} 16.36 \\ -5.90 \end{bmatrix}, \; K^{d\dagger} = \begin{bmatrix} -5.48 & 1.98 \\ -11.76 & 3.90 \end{bmatrix}.$$

Based on these target policies, solving the optimization problem (19) gives the attacker the following cost parameters, which are used to generate the poisoned dataset and achieve its ultimate objective:

$$\tilde{Q} = \begin{bmatrix} 1.21 & 0.32 \\ 0.15 & 1.41 \end{bmatrix}, \; \tilde{R} = 1.65, \; \tilde{M} = \begin{bmatrix} 4.9 & 0 \\ 0.54 & 4.6 \end{bmatrix}.$$

When the players estimate the system and cost parameters from the manipulated dataset, they arrive at the following policies, which adhere to the adversary's objective:

$$\hat{K}^{u} = \begin{bmatrix} 15.53 \\ -5.60 \end{bmatrix}, \; \hat{K}^{d} = \begin{bmatrix} -4.62 & 1.69 \\ -11.74 & 3.89 \end{bmatrix}.$$

The trajectories of the two players when using the original and poisoned dataset are shown in Fig. 1, which can be understood as estimates of K^* in blue vs. $K^\dagger$ in red. It is observed that the attacker successfully tricks the players into following less cost-effective trajectories, with the trajectories of Player 1 and Player 2 affected to a similar degree. Fig. 2 illustrates the magnitude of the changes to the dataset performed by the attacker, demonstrating that the attack is effectively carried out with minimal changes.

B. Pursuit-Evasion Game

In this case study, we examine the attack model occurring within a pursuit-evasion game.

A pursuit-evasion LQ differential zero-sum game is a problem classified by the countering strategies of targeting and avoidance [16]. The cost is the distance between the two subjects over time. In this case, it is more likely that the malicious policy will favor one player over the other due to the asymmetry of player objectives.

Let $y_p \in \mathbb{R}^2$ define the position of the pursuer and $z_p = \dot{y}_p \in \mathbb{R}^2$ its velocity. Let $x_p = [y_p^T, z_p^T]^T$ define the state vector of the pursuer with dynamics controlled by its input $u_p \in \mathbb{R}^2$, representing the acceleration force. The state dynamics of the pursuer are:

$$\dot{x}_p = Ax_p + B_p u_p,$$

where

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \otimes I_2, \; B_p = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes I_2.$$

Similarly, $y_e \in \mathbb{R}^2$ and $z_e = \dot{y}_e \in \mathbb{R}^2$ represent the position and velocity vectors of the evader, and $x_e = [y_e^T, z_e^T]^T$ define the state vector, and the control is defined by $u_e \in \mathbb{R}^2$. The state dynamics of the evader can be understood as:

$$\dot{x}_e = Ax_e + B_e u_e,$$

with

$$B_e = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes I_2.$$

Defining a new state $x := x_p - x_e$, the dynamics of the pursuit-evasion game can be understood as:

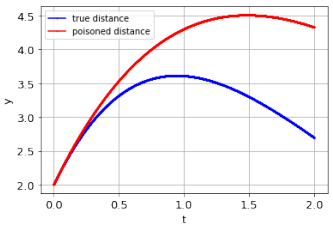
$$\dot{x} = Ax + B_p u_p - B_e u_e.$$

Denote by $x := (x_1, x_2, x_3, x_4)$, we obtain the following representation:

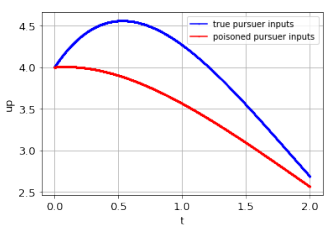
$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} u_p - \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} u_e.$$

the sampling interval is defined as $\Delta t = 0.0001$, the cost matrices as: $Q = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \otimes I_2, R = I_2, M = 2I_2$ and the initial positions and velocities of the pursuer and evader as $x_p(0) = [-5, -2, 5, 1]^T$, $x_e(0) = [5, 10, 1, 1]^T$.

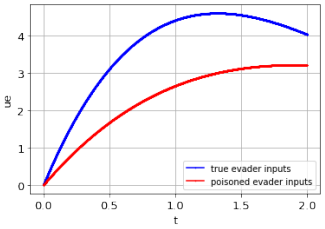
The attack is performed similarly to the case study in Sec. IV-A. The differences in the game state between the policy pairs obtained for the clean and poisoned datasets as well as the differences in the resulting inputs of the pursuer and evader



(a) Euclidean distance between pursuer and evader

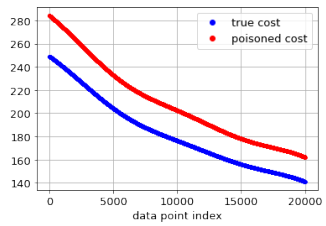


(b) Pursuer's adjustments

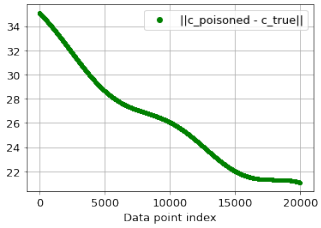


(c) Evader's adjustments

Fig. 3. (a): The differences in the game state resulting from the policy pairs obtained from the clean and poisoned datasets, in blue and red respectively. (b), (c): The differences in resulting inputs between the clean and poisoned datasets for the pursuer and evader, respectively.



(a) Cost values



(b) Attack cost

Fig. 4. (a): Cost observations in the original batch dataset and poisoned dataset, in blue and red respectively, at each data point index. (b): The attack cost defined as the difference between the two cost observations, in green.

are displayed in Fig. 3. In Fig. 3(a), the evader is getting away from the pursuer. We expect the pursuer to catch up to the evader's velocity and start shrinking the distance starting at around $t = 1$. It is seen that the data poisoning attack causes the distance between the pursuer and the evader not to shrink until $t = 1.5$, a delay of 0.5. Thus, this attack favors the evader. In Fig. 3(b) and Fig. 3(c), both the pursuer and the evaders are shown to use weaker input signals in the poisoned case, so the attack does not favor either player in terms of control inputs.

The attacker's changes to the batch dataset can be observed in Figure 4. Fig. 4(a) shows the cost values in the true batch data in blue and the poisoned data in red, with minimal differences between them. Further, Fig. 4(b) shows the attack cost, defined as the difference between the two datasets at each point, quantifying the effectiveness of the attack with the amplified deceptive behavior displayed in Fig. 4(a).

V. Conclusion

In this study, we crafted a strategic attack model to interfere in two-player LQ zero-sum differential games that learn their feedback control policies from a batch dataset. Using the proposed model, an adversary is able to guide the players to its own target policies in manipulating the cost measures of the dataset, and accomplish this minimally to avoid detection. Future studies may explore the manipulation of other measurements in the batch data and extend the model to an N-player game. It is also essential to investigate means to detect minimal changes in data and defend from attacks in these contexts, such as cryptographic techniques to ensure data integrity.

References

[1] Y. Kim and H. Bang, "Introduction to Kalman filter and its applications," *Introduction and Implementations of the Kalman Filter*, vol. 1, pp. 1–16, 2018.

[2] M. Palan, S. Barratt, A. McCauley, D. Sadigh, V. Sindhwani, and S. Boyd, "Fitting a linear control policy to demonstrations with a Kalman constraint," in *Learning for Dynamics and Control*. PMLR, 2020, pp. 374–383.

[3] R. Isaacs, *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. Courier Corporation, 1999.

[4] J. H. Fullard, J. A. Simmons, and P. A. Saillant, "Jamming bat echolocation: the dogbane tiger moth cycnia tenera times its clicks to the terminal attack calls of the big brown bat eptesicus fuscus." *The Journal of experimental biology*, vol. 194, no. 1, pp. 285–298, 1994.

[5] Y. Huang and Q. Zhu, "Deceptive reinforcement learning under adversarial manipulations on cost signals," in *Decision and Game Theory for Security*. Springer, 2019, pp. 217–237.

[6] 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, 2020.

[7] C. M. King, S. T. Do, and J. Chen, "Policy poisoning in batch learning for linear quadratic control systems via state manipulation," in *57th Annual Conference on Information Sciences and Systems (CISS)*, 2023, pp. 1–6.

[8] 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.

[9] P. Zhang, "Some results on two-person zero-sum linear quadratic differential games," *SIAM Journal on Control and Optimization*, vol. 43, no. 6, pp. 2157–2165, 2005.

[10] Y. Huang, J. Chen, and Q. Zhu, "Defending an asset with partial information and selected observations: A differential game framework," in *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021, pp. 2366–2373.

[11] Y. Huang and Q. Zhu, "A pursuit-evasion differential game with strategic information acquisition," *arXiv preprint arXiv:2102.05469*, 2021.

[12] T. Başar and G. J. Olsder, *Dynamic Noncooperative Game Theory*. SIAM, 1998.

[13] J. Engwerda, *LQ Dynamic Optimization and Differential Games*. John Wiley & Sons, 2005.

[14] 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.

[15] 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.

[16] B. Al Faiya, "Learning in pursuit-evasion differential games using reinforcement fuzzy learning," Ph.D. dissertation, Carleton University, 2012.