Towards Physics-Informed Diffusion Reinforcement Learning for Investigating Adversarial Trajectories

Arun Sharma[1]

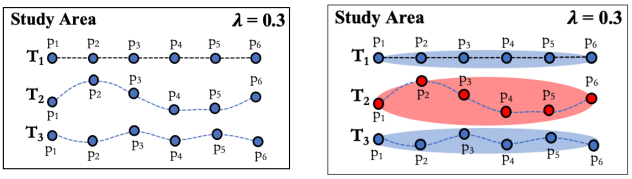
[1]Department of Computer Science, University of Minnesota, Twin Cities, USA
{arunshar}@umn.edu

Abstract

Given trajectory data, a domain-specific study area, and a user-defined threshold, we aim to find anomalous trajectories indicative of possible deception-based activity (i.e., an activity that attempts to conceal its movements by intentionally emitting false signals). The problem, especially in the maritime domain, is societally important to curb illegal activities in international waters, such as unauthorized fishing and illicit oil transfers. The problem is computationally challenging due to a potentially infinite variety of fake trajectory behavior and a lack of ground truth to train state-of-the-art generative models. Due to data sparsity, prior approaches capture anomalous trajectories with high false positive rates, resulting in lower accuracy. To address these limitations, we propose a physics-informed diffusion model that integrates kinematic knowledge to identify trajectories that do not adhere to physical laws. Experimental results on real-world datasets in the maritime and urban domains show the proposed framework provides higher prediction accuracy than baseline generative models.

Introduction

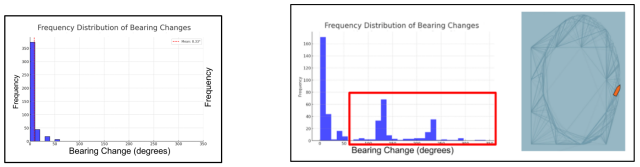
Given trajectory data, a domain-specific study area (e.g., road networks, maritime waters), and a user-defined reconstruction threshold, our goal is to identify anomalous trajectories that indicate deception-based behavior. Deception-based anomalies arise when an entity deliberately alters or falsifies its reported locations to obscure its true movement, often as part of illicit activities such as unauthorized vessel movement, illegal fishing, or fraudulent positioning in GPS-based systems. These deceptive signals deviate from expected traffic patterns and violate fundamental physical constraints, leading to higher reconstruction errors in traditional generative models. For instance, as shown in Figure 1a, trajectories T_1, T_2, T_3 illustrate different mobility behaviors, where T_2 exhibits abnormal motion characteristics compared to T_1 and T_3, particularly in terms of angular deviation and physical constraints such as speed. In Figure 1b, T_2's behavior is flagged as anomalous due to its deviation from historical trajectory patterns and violation of physical feasibility constraints (e.g., bearing, acceleration), as reflected by its reconstruction error exceeding 0.3.



(a) Trajectory Dataset (b) Abnormal Trajectory

Figure 1: An illustration example of problem statement where trajectories T_1, T_2, T_3 are input (left) and T_2 is classified as abnormal with reconstruction threshold $\lambda \geq 0.3$

To address these challenges, we propose a **Physics-informed Reinforcement Learning Model (Pi-RL)** that combines reinforcement learning with kinematic bicycle motion constraints to effectively analyze spatiotemporal characteristics of trajectories. By leveraging **Deep Q-Networks (DQN)** for anomaly detection and **Proximal Policy Optimization (PPO)** for trajectory modeling, Pi-RL learns the difference between physically valid and deceptive movements, ensuring that detected anomalies reflect genuine deviations from motion laws rather than statistical outliers. Experimental results on real-world datasets across urban and maritime domains demonstrate that our approach provides superior anomaly detection accuracy compared to state-of-the-art generative models while significantly reducing false positive rates.



(a) Normal Behavior (b) Abnormal Behavior

Figure 2: A case study of an oil tanker Cathay Pheonix whose bearing distribution (right) is significantly different towards normal behavior (left).

Detecting anomalous trajectories is critical for applications such as autonomous driving, maritime monitoring, and security surveillance. In particular, *deception-based anoma-*

lies – where an agent intentionally broadcasts false location signals to conceal its true trajectory pose a serious challenge (Liu et al. 2024). These anomalies are difficult to detect due to the infinite ways an adversary can generate fake yet plausible trajectories and the scarcity of labeled examples for training.

Limitations of Related Work: Traditional trajectory anomaly detection methods often rely on statistical or machine learning models to learn "normal" movement patterns and flag deviations. However, purely data-driven models may struggle to distinguish true anomalies from benign outliers, especially when they ignore the underlying physics of motion. *Physics-informed methods* incorporate known physical constraints, such as the kinematic bicycle model (KBM), to improve detection robustness (Kong et al. 2015; Polack et al. 2017). The KBM provides a simplified yet effective representation of vehicle motion by capturing key kinematic relationships between velocity, heading, and steering, making it widely used for path planning (Ziegler, Willert, and Adamy 2022). Recent works have successfully integrated KBM into deep learning frameworks to enhance trajectory modeling and improve interpretability in anomaly detection (Liu et al. 2020; Zhang et al. 2023). This fusion of physics-based modeling and learning is particularly advantageous in scenarios where deceptive trajectory modifications must be identified while maintaining consistency with realistic motion constraints.

In parallel, *Reinforcement Learning (RL)* has emerged as a powerful framework for sequential decision-making, achieving remarkable success in tasks ranging from Atari games (Mnih et al. 2015) to continuous control applications (Schulman et al. 2017). Deep RL algorithms such as Deep Q-Networks (DQN) and Proximal Policy Optimization (PPO) enable agents to optimize policies over sequential interactions (Hessel et al. 2018; Schulman et al. 2015). Unlike supervised learning, RL does not require explicit labels and instead learns from rewards, making it well-suited for anomaly detection over trajectory data. Specifically, RL can be used to either *optimize trajectory following*—by training agents to imitate normal behavior—or *search for anomalies* by formulating detection as a sequential decision problem (Zhang et al. 2023; Alsabah et al. 2019). Furthermore, *model-based RL* methods leverage predictive dynamics models, such as the KBM, to plan ahead and enforce physical consistency, which can improve sample efficiency and robustness in detecting deceptive trajectory modifications (Chua et al. 2018; Janner et al. 2019). This work builds on these advances by integrating RL with physics-informed modeling to enhance anomaly detection in trajectory data.

Contributions: This paper makes three key contributions:

- We introduce a **Physics-Informed RL Framework** for trajectory anomaly detection, integrating the kinematic bicycle model (KBM) to ensure learned policies adhere to vehicle motion constraints, improving interpretability and robustness.
- We apply and compare **model-free (DQN, PPO) and model-based RL** for detecting anomalous trajectories, leveraging KBM dynamics for enhanced performance.
- Our approach effectively detects **deception-based anomalies** by identifying physically inconsistent trajectories that evade statistical detectors.
- We implement and evaluate the RL+KBM framework on simulated and real-world datasets, demonstrating superior accuracy, reduced false alarms, and improved anomaly detection compared to baselines.

Organization: The remainder of this paper is organized as follows. Section 2 reviews relevant background in reinforcement learning and trajectory anomaly detection. Section 3 details our methodology, including the RL formulation, integration of the kinematic bicycle model, and handling of deceptive anomalies. Section 4 describes the implementation and experimental setup, while Section 5 presents results and evaluations of the proposed approach. Section 6 provides a discussion of the findings, and Section 7 concludes the paper with final remarks and future directions.

Problem Formulation

Notations and Definitions

Definition 1. *A **location trace** p represents a broadcasted signal from a GPS or Automatic Identification System (AIS) transponder, carrying information such as longitude (lon), latitude (lat), timestamp (t), and velocity (v).*

*A **spatial trajectory** is a collection of location traces generated by a moving object, typically represented as a chronologically ordered sequence of points, $p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_n$, where each point p consists of $(lat, long, t, v)$. For instance, in Figure 1a, the trajectories T_1, T_2, and T_3 consist of ordered location traces, where each trajectory T_i is defined as $[p_1, \ldots, p_n]$. A collection of such trajectories forms a trajectory dataset, denoted as $T = [T_1, T_2, \ldots, T_n]$.*

Definition 2. *A **reconstruction threshold** λ is the error margin derived from comparing the divergence between reconstructed and original input using a generative model.*

A trajectory is classified as an **anomalous trajectory** T_a if its reconstruction error exceeds or equals λ, indicating significant deviations in lat/lon, speed, or heading compared to neighborhood trajectories. For instance, in Figure 1b, trajectory T_2 in red exhibits substantial deviations relative to T_1 and T_3, with a reconstruction error ≥ 0.3, and is therefore flagged as anomalous.

Problem Statement

Input:

- A trajectory dataset T consisting of multiple trajectories $T = \{T_1, T_2, ..., T_n\}$.
- A reinforcement learning-based anomaly detection model integrating:
  - **Deep Q-Network (DQN):** for discrete anomaly decision-making.
  - **Proximal Policy Optimization (PPO):** for trajectory optimization.
  - **Kinematic Bicycle Model (KBM):** enforcing physical motion constraints.
- A **reconstruction threshold** λ to quantify deviation from expected motion behavior.

Output:

- A set of anomalous trajectories T_a where the reconstruction error is $\geq \lambda$.

Objective:

- Detect deceptive or physically implausible trajectories using RL and KBM constraints.
- Improve anomaly detection accuracy by leveraging physics-informed reinforcement learning.
- Optimize solution quality in terms of recall, precision, F1-score, and AUROC.

Constraints:

- Limited availability of labeled anomalous trajectories.
- Real-time anomaly detection in streaming data.
- Ensuring learned policies respect physical constraints.

Related Work

Deep Q-Networks (DQN). Deep Q-Networks were a breakthrough in deep reinforcement learning, combining Q-learning with deep neural networks and experience replay to learn value-based policies from high-dimensional inputs. In the seminal work by Mnih et al. (2015), a single DQN agent achieved human-level control on dozens of Atari 2600 video games directly from raw pixel inputs. DQN approximates the optimal action-value function $Q^*(s, a)$ using a neural network, and updates this network by minimizing the temporal-difference error between predicted Q-values and target values obtained from reward plus the highest next-state Q (with some stabilization techniques like fixed target networks and replay memory). The success of DQN demonstrated that *value-based* deep RL can learn complex policies end-to-end; for instance, DQN learned strategic gameplay such as tunneling behind bricks in Breakout to maximize long-term rewards. In the context of trajectory analysis, a DQN could be applied if we discretize the action space (e.g., discretizing steering or classification decisions) and use a suitable state representation. Notably, DQN (and its variants like Double DQN and Dueling DQN) has been extended beyond games to control problems, although continuous control often favors policy gradient methods. For anomaly detection, value-based RL can be used to learn a *Q-value anomaly score* for trajectory segments – for example, an agent could learn the expected "normality" value of being in a given trajectory state, and flag low-valued states as anomalous. Zhang et al. (2022) took a related approach in *RL4OASD*, an RL-based online anomaly detector for trajectories, where one network learns trajectory features and another (potentially value-based) network identifies anomalous sub-trajectories without requiring supervised labels.

Proximal Policy Optimization (PPO). Proximal Policy Optimization is a widely used model-free RL algorithm in the actor-critic family, introduced by Schulman et al. (2017) as an improvement over Trust Region Policy Optimization (TRPO). PPO uses a *policy gradient* approach with a clipped surrogate objective to ensure stable and monotonic policy updates without requiring second-order derivatives. In essence, PPO constrains the new policy not to deviate too far from the old policy during each update by clipping the probability ratio, which balances exploration and stability. PPO has become popular for continuous control tasks (robotics, autonomous driving simulations, etc.) due to its simplicity, sample efficiency, and robustness to hyperparameters. In autonomous driving research, PPO and similar continuous-action RL algorithms (like DDPG, TD3, SAC) are commonly used to train agents to control vehicles (steering, acceleration) in simulators. For example, an RL agent controlling a car might use throttle and steering as continuous actions and receive rewards for staying in lane or reaching a destination safely. When using PPO for trajectory anomaly detection, one approach is to train an agent to *imitate normal driving behavior*: the reward can be designed to encourage following the road network, obeying speed limits, and minimizing abrupt changes. The agent (policy) thus captures the distribution of normal trajectories. If we then evaluate a new trajectory under this learned policy or reward function, anomalies can be identified by low reward or large deviations. Prior work on *intention-oriented anomaly detection* has leveraged *inverse* reinforcement learning to recover a reward function for normal behavior, then used that to detect anomalous trajectories that do not maximize the learned reward. Here, we use forward RL (like PPO) to directly learn a policy for normal movements; this policy can serve as a reference model to detect deviations. Additionally, PPO can be used in an adversarial training setting, where one agent tries to follow normal physics and another agent tries to introduce deceptive perturbations – PPO's stability is advantageous in such two-player training.

Model-Based RL and Physics Integration. In model-free RL, the agent learns solely from experience, treating the environment as a black box. *Model-based reinforcement learning (MBRL)*, on the other hand, uses an internal model of the environment's dynamics $f(s, a)$ to plan or generate additional simulated experiences. If a reliable model of the system is known (or learned), MBRL can dramatically improve sample efficiency by enabling the agent to anticipate outcomes or perform lookahead search. The domain of autonomous vehicles is well-suited to model-based approaches because the physics of vehicle motion are reasonably well understood. In fact, optimal control methods in robotics and vehicles (like Model Predictive Control) have long used known dynamics models for planning. The *kinematic bicycle model* is a prime example of a simple yet effective dynamics model for cars: it assumes a simplified geometry (two-wheel model, no tire slip, valid at low to moderate speeds) and uses vehicle velocity and steering angle (or curvature) to predict how the vehicle's position and orientation evolve. As Georgiev (2023) noted, if our system (a car) is well understood, we have decades of research to derive a physics model; the KBM provides such a model by lumping a car's wheels into a bicycle-like abstraction with parameters like wheelbase and steering angle. By incorporating the KBM into RL, we essentially equip the agent with domain knowledge of driving physics. Model-based RL can use the KBM in several ways: (1) as the actual environment model (i.e., the simulation step for the agent is computed by KBM equations, which we do in this work), or (2) as a *learned model regularizer*, where the agent's learned transition function is constrained to match KBM predictions (this could be part of the loss function in a differentiable

planning network). In our approach, the KBM is directly used for state transition, so the agent never explores physically impossible transitions. This not only makes learning more efficient (fewer irrelevant states explored) but also aids anomaly detection: any trajectory data that significantly violates the KBM dynamics will appear surprising to the agent. Prior works have successfully combined physics models with learning for trajectory generation; for instance, Chen et al. (2023) guide a neural stochastic differential equation with a bicycle model to ensure generated trajectories are physically feasible. Similarly, our use of the KBM in RL serves to *ground the learning in real-world physics*, aligning with the concept of physics-informed machine learning.

RL for Trajectory Anomaly Detection. Using RL specifically for anomaly detection is a relatively recent development. A key challenge is that anomaly detection is often an *unsupervised or semi-supervised* problem – we may not have reward signals readily available for "detecting an anomaly," since true anomalies are rare and not labeled. Researchers have addressed this by crafting surrogate reward functions or training schemes. Zhang et al.'s RL4OASD (mentioned above) avoids needing labeled anomalies by training two networks in tandem: one learns a representation of trajectories (state embedding) and the other is an RL-based detector that receives rewards for identifying unlikely subtrajectories. The reward can be derived from a statistical rarity measure or reconstruction error provided by the first network. In essence, the RL agent is encouraged to pinpoint segments of a trajectory that do not fit the learned normal profile. This approach significantly outperformed earlier methods (20–30% improvement in detection accuracy) on real-world taxi trajectories, demonstrating the efficacy of RL in this domain. Another approach is formulating anomaly detection as a sequential decision process where an agent scans through streaming trajectory data and must decide whether to raise an alarm (and where). The agent gets penalized for false alarms or missed detections, and rewarded for correct, timely detection. This framing can be tackled with RL algorithms (e.g., a DQN that at each step chooses between "continue" or "flag anomaly"). Additionally, *adversarial reinforcement learning* can be employed: one agent generates trajectories (possibly trying to mimic normal ones but with subtle anomalies) and another agent learns to detect them, with both evolving their strategies. Such adversarial setups force the detector to become more robust to tricky, deceptive anomalies.

Proposed Approach

In this section, we outline our proposed *Reinforcement Learning with Kinematic Bicycle Model (RL+KBM)* framework for trajectory anomaly detection. We first describe the overall approach and how we formulate anomaly detection as an RL problem. We then detail the integration of the kinematic bicycle model into the RL environment and learning process, enabling physics-informed policy optimization. Next, we discuss how specific RL techniques (DQN, PPO, and a model-based strategy) are applied within this framework for trajectory optimization and anomaly identification. Finally, we explain our strategy for handling deception-based anomalies, wherein the RL agent is trained to recognize and adapt to intentionally falsified trajectory data.

State and Action Spaces: We consider an environment representing an agent (e.g., a vehicle or vessel) moving in a 2D plane. The *state* s_t at time t contains the agent's kinematic state, which we define based on the KBM: $s_t = (x_t, y_t, \theta_t, v_t)$, where (x_t, y_t) is the position, θ_t is the heading (orientation), and v_t is the speed. Additional state features can include past positions or contextual info (like road lane or traffic conditions), but for the core dynamics we use these four variables

The *action* a_t consists of the control inputs that change the state. In a standard rear-axle KBM, the controls could be acceleration (or directly velocity command) and steering angle rate

To simplify, we parameterize actions as $a_t = (\Delta v_t, \Delta\kappa_t)$, where Δv_t is the change in velocity (acceleration/braking) and $\Delta\kappa_t$ is the change in curvature. Here, curvature $\kappa = \tan(\delta)/L$ is used instead of steering angle δ so that our model is not tied to a specific wheelbase

This means the agent's actions effectively adjust its speed and turning curvature at each step.

State Transition (Environment Dynamics): The environment simulates the agent's motion using the kinematic bicycle model equations. Given state s_t and action a_t, the next state s_{t+1} is computed by integrating the KBM ODEs (either analytically for a small time Δt or via Euler integration for simplicity). For example, using a time step Δt:

$$x_{t+1} = x_t + v_t \cos(\theta_t)\,\Delta t, \quad y_{t+1} = y_t + v_t \sin(\theta_t)\,\Delta t,$$

$$\theta_{t+1} = \theta_t + v_t \tan(\delta_t)/L\,\Delta t, \quad v_{t+1} = v_t + \Delta v_t.$$

The steering angle δ_t (if used) would be updated as $\delta_{t+1} = \delta_t + \Delta\delta_t$ (with $\Delta\delta_t$ bounded by some max steering rate). In our formulation, using curvature κ, we directly update κ: $\kappa_{t+1} = \kappa_t + \Delta\kappa_t$. The curvature influences the heading change as shown above.

These equations ensure the agent's trajectory is smooth and feasible: no teleportation or instantaneous heading changes can occur beyond what the kinematics allow.

Reward Design: Designing the reward function $r(s, a)$ is crucial, as it drives the RL agent's behavior. For anomaly detection, we consider two complementary reward designs:

Imitation of Normal Trajectory Behavior: Here, we reward the agent for following a trajectory that appears normal or typical. If we have a set of example normal trajectories (from real data or from a path planning algorithm), we can reward the agent for staying close to those trajectories or achieving similar motion patterns. For instance, the reward could include terms like `-distance` to a reference path, `-deviation` from typical speed, and penalties for excessive acceleration or turn rates. The agent trained with this reward will learn a policy $\pi(s)$ that generates realistic, non-anomalous trajectories. At test time, we can use this learned policy to evaluate new trajectories: if following the trajectory yields large negative reward (meaning the trajectory is

hard to follow with the normal policy) or the trajectory diverges from what the policy would do, then the trajectory is likely anomalous.

Direct Anomaly Signaling: Alternatively, we can include an explicit anomaly flag action in the agent's action space and shape rewards to directly incentivize correct detection. For example, the agent could have a binary action at each step: either continue normal operation or declare an anomaly. We can simulate various trajectories, some normal and some containing anomalies, and give a positive reward if the agent declares anomaly in an anomalous scenario (and a penalty if it fails to) and vice versa for normal scenarios (penalize false alarms). However, this requires either known anomalies or a way to simulate anomalies during training. Given the unsupervised nature, a more feasible approach is to combine this with the first method: the agent learns to *expect* normal behavior, and we generate anomalies in simulation (like injecting sudden jumps in the trajectory). The agent then gets a large negative reward whenever the observed state deviates from what the KBM and the agent's policy predict (since that indicates an anomaly). In effect, the agent can learn to *raise an alarm* when its prediction error exceeds a threshold.

In our framework, we lean towards the first method (learning normal behavior) with an implicit anomaly detection by monitoring reward or error, but we also incorporate aspects of the second via *curiosity or error-based reward*. Specifically, we include an intrinsic reward for the agent that is the *negative prediction error* between the expected next state (according to the KBM and policy) and the actual observed next state. Under normal conditions, this error is just process noise; under a deception anomaly (where the reported position is fake), this error spikes, effectively penalizing the agent and signaling something is off. This technique is inspired by approaches in which RL agents use prediction error as a signal for novelty or anomaly (sometimes called curiosity-driven learning in other contexts).

Episode Design: Each episode in the RL training represents a trajectory segment. We initialize the agent in some state (either sampled from real data distribution of normal trajectories or a random plausible state). The agent then takes a sequence of actions according to its policy and the environment updates the state using KBM. For training on normal behavior, the environment could also secretly follow a reference normal trajectory and include a guiding reward to move toward the next point on that reference (this is a form of *trajectory following* task). An episode might end after a fixed number of steps or when the agent deviates too far from valid bounds (e.g., leaving the road or exceeding speed limits, which might be considered failure). For training with anomaly signals, we might end an episode early if a clear anomaly is detected or if the agent correctly flags an anomaly. We also inject random *anomaly events* in some training episodes (like at a random time step, we teleport the reported position or alter the heading drastically to simulate a fake signal) so that the agent experiences both normal and abnormal transitions and can learn to identify the latter.

1. Integrating the Kinematic Bicycle Model into RL

A core aspect of our methodology is embedding the *Kinematic Bicycle Model* into the RL loop, making the learning process *physics-informed*. We achieve this at multiple levels:

Environment Model: As described, the environment transitions are computed using the KBM equations. This means the RL agent is effectively interacting with a *simulator* that is grounded in real vehicle physics. Unlike a generic gridworld or a learned neural network environment, here the next state is determined by known formulae. This has an immediate advantage: we do not need to learn the basic physics from data – it is given. The agent thus can focus on higher-level pattern learning (e.g., what sequence of turns and moves constitutes normal driving on a highway versus abnormal weaving).

State Feature Engineering: The state representation can explicitly include physical quantities derived from the KBM. For example, we can include the heading θ and speed v directly (rather than implicitly having to infer them from raw coordinates). We can also provide the agent with its control outputs as part of the state if helpful (though in a Markov decision process this is not strictly necessary). Another idea is to include *physical consistency checks* as part of the state or reward – e.g., compare the agent's estimated position (integrating its known controls) with an observed position (which could be noisily measured). Any large discrepancy could be fed as an input to the agent, essentially alerting it to a violation of the model.

Physics-Based Reward Shaping: We incorporate terms in the reward that reflect adherence to physical laws. For instance, we penalize the agent for actions that would push the dynamics to extremes (very high Δv or $\Delta \kappa$ beyond plausible human driving patterns). We also add penalties for *jerk* or sudden changes. This encourages smoother trajectories that align with physical expectations. Such shaping not only yields more realistic behavior but also teaches the agent what "normal" physics-constrained motion looks like, so that encountering a physically inconsistent situation is inherently out-of-distribution for the agent.

Model-Based Planning: Although we use the KBM as the ground-truth environment, we can additionally leverage it for internal planning within the agent (a model-based RL step). For example, a DQN agent can use *Dyna-Q*-style updates: it can take experienced transitions (s, a, s') and feed them into a model (which in this case is the same KBM equations) to hallucinate additional transitions or to simulate different action outcomes from state s (since the model is known and differentiable). Similarly, a PPO agent could use the model to simulate trajectories when computing the policy loss or value function targets beyond the actual rollout (this is related to *imagination rollout* in algorithms like MBPO or Dreamer). Because the KBM is computationally cheap to simulate, we can afford to generate many hypothetical scenarios for the agent to practice on, improving data efficiency. In our implementation, after each real episode, we let the agent do extra "mental rehearsal" by picking some states it saw and simulating random action sequences with the KBM to see the outcomes – updating its value network with these imagined trajectories as if they were real. This

is a classic model-based augmentation to training that helps the agent converge faster and potentially anticipate out-of-distribution situations.

Constraint Enforcement: In addition to guiding the agent, the KBM can be used to enforce hard constraints. For example, if the task demands that the agent never exceed a certain lateral acceleration (a safety limit), we can derive that from the KBM (lateral acceleration $\approx v^2 \tan(\delta)/L$) and simply not allow actions that break it. Or if an anomaly is defined by violating a physical limit, we can automatically label such states as anomalous. During training, if the agent somehow attempts an action outside the feasible set (due to function approximation issues or exploration), the KBM can either clip that action to the nearest valid or terminate the episode as a failure. This keeps the policy within the realm of feasible physics at all times.

By tightly coupling the physics model with RL, our approach ensures that the learned policy $\pi(s)$ and value function $V(s)$ have a strong notion of what is physically plausible. This not only improves the realism of trajectory optimization but also gives a *built-in anomaly detector*: any real trajectory that the policy finds inexplicable (in terms of high error or low value) under the KBM likely contains an anomaly. In other words, the difference between what the physics-informed RL agent expects and what actually occurs can be used as an anomaly score.

2. Integrating DQN and PPO in RL+KBM

We implement two model-free RL agents within our framework to cover both discrete and continuous decision paradigms:

(i) DQN Agent for Anomaly Identification: The DQN agent treats anomaly detection as a discrete action problem. At each decision step (which could correspond to each new observed trajectory point or at key points along the path), the agent chooses among a discrete set of actions. These actions could be, for example: {0: continue normal, 1: flag anomaly} in a simple detection setting, or a slightly richer set like {0: follow/reference action, 1: investigate anomaly} where "investigate" might correspond to taking a different control action to test the system. In our design, we use a binary action: either *no anomaly* or *anomaly detected now*. The DQN's state input not only includes the kinematic state (x, y, θ, v) but also a representation of the trajectory history (e.g., recent states or a recurrent encoding) and the discrepancy between expected and observed position if any. The DQN network then outputs a Q-value for each action. A high Q for "anomaly" would mean the agent believes declaring an anomaly now yields high future reward (which would be set to correlate with true positive detection). To train this, we generate episodes where some contain anomalies. The reward scheme is as follows:

- A large positive reward is given if the agent correctly flags an anomaly (and episode ends).
- A large negative reward if it fails to flag by the end of an anomalous episode.
- A small negative step penalty for each time step to encourage prompt decisions.
- For normal episodes (no anomaly), the agent should ideally never flag (we give negative reward if it does, and zero if it never flags by end).

We use the standard DQN training with experience replay and ϵ-greedy exploration. Transitions are of form (s, a, r, s') with the terminal state when anomaly is flagged or end of trajectory. The DQN learns to approximate the action-value function for these decisions. Over time, it will learn to recognize the telltale signs of an anomaly in the state (for instance, if the observed motion deviates from what the kinematic model would predict given the agent's last action, the state will contain a large error signal, and DQN will learn that "anomaly" action yields a high reward in such states). One challenge is that anomalies are rare and initially the agent may not experience them often; we address this by oversampling anomalous scenarios during training and by an *abnormality bonus* – i.e., a slight intrinsic reward whenever the prediction error is high, to draw the agent's attention to those states.

(ii) PPO Agent for Trajectory Imitation: The PPO agent operates in a continuous action space, directly outputting control actions $(\Delta v, \Delta\kappa)$ at each time step, aiming to *follow normal trajectories*. This agent's primary goal is not to declare anomalies, but to smoothly control the vehicle. We train it on many example trajectories that are deemed normal (these could be from a dataset of real drivers or from a motion planner). The reward for PPO at time t could include: (a) a *tracking reward* that is highest when the agent's state matches the next point of a reference trajectory (if one is provided for that episode), (b) a small penalty on control effort to encourage smooth driving, and (c) a penalty for deviations from road/lane (if context is included). Additionally, we inject occasionally an episode where the agent's sensor input (the observed state) is tampered to simulate deception. For example, the agent might be on a straight road but the observed position jumps to a location off-road. The agent, following its policy, might output a drastic control to correct, but since the environment is actually using the true state (if we simulate the truth vs observed), we can measure that the agent was "confused." In such cases, we provide a negative reward spike to mark that something undesirable happened (the agent's belief and reality diverged). The PPO algorithm optimizes the policy to maximize the expected cumulative reward

Since the majority of training data is normal, PPO will learn a policy that replicates normal behavior. Importantly, PPO also learns a *value function* $V(s)$ (the critic) estimating the expected return from state s. We hypothesize that $V(s)$ can serve as an *anomaly score*: if the agent finds itself in a state that it thinks will lead to poor returns (because maybe it senses things have gone awry), the value will drop. Deceptive anomalies that suddenly move the agent to a weird state will likely cause a low value. In deployment, we thus monitor the PPO agent's value predictions or its action outputs when tracking an observed trajectory: if the agent either cannot track the trajectory (e.g., its actions saturate) or the predicted value plummets, we flag an anomaly.

Both the DQN and PPO agents benefit from the *KBM in-*

tegration. For DQN, the state includes features derived from the KBM (like the predicted next state vs. actual). For PPO, the entire training environment is using KBM, so the policy inherently knows how a normal trajectory should evolve physically. We found that a combination of both approaches can be useful: PPO as a *trajectory replicator* to generate expected behavior and DQN as a *decision module* to signal anomalies. In a combined system, the PPO agent can run in parallel with a tracking filter to continuously predict where the object should be, while the DQN agent takes as input the current state and the PPO's prediction error to decide if this is an anomaly. This resembles how an autopilot (predictive model) and a monitoring system might work together.

(iii) Model-Based RL Enhancements

In addition to using the KBM as the environment, we experimented with fully *model-based RL planning*. One approach is to use *Monte Carlo Tree Search (MCTS)* or rollout simulation at each step to explicitly evaluate the consequences of declaring an anomaly versus not, under the known dynamics. Because anomaly declaration ends the episode (and yields certain reward if correct or penalty if wrong), we can simulate forward to see if staying silent leads to a state that is very unlikely under the model. If so, it might be better to declare. MCTS can effectively look a few steps ahead using the KBM (and some heuristic model of how an anomaly might continue). However, performing MCTS at each time step can be computationally expensive in a streaming scenario, so in our implementation we rely more on the learned policies (which approximate these decisions) rather than online search. Still, during training, we use the model to generate *imagined trajectories*: e.g., starting from an anomalous state, simulate what a normal agent would do and see the discrepancy. These imagined outcomes enrich the replay buffer for DQN or help compute advantage estimates for PPO beyond what was actually observed.

Another facet of model-based reasoning is *system identification*. The KBM has parameters like wheelbase L or in ships a rudder delay, etc. If our agent encounters a new object (vehicle type) for which these parameters differ, its model predictions might systematically err. This is not an anomaly of the type we want (not deception, just model mismatch). To handle this, we include a mechanism to adapt the model: the RL agent can periodically update the parameters using observations (like estimating the effective turning radius by comparing successive positions). This can be done via a secondary learning process (perhaps a supervised regression or a simple least-squares fit on a window of data). By keeping the model tuned, we ensure that only genuine anomalies (not just modeling errors) are flagged. This combination of model-based adaptation and reinforcement learning is aligned with ideas in adaptive control and meta-RL, but our focus remains primarily on detection rather than control optimality.

(iv) Handling Deception-Based Anomalies

Deception-based anomalies are those where an agent deliberately emits false trajectory data to deceive observers. Examples include a ship spoofing GPS coordinates or an autonomous vehicle reporting back incorrect telemetry. These anomalies are particularly challenging because the false data may be *physically plausible* at first glance – the adversary may generate lies that still follow basic motion constraints to avoid easy detection. Our RL+KBM approach tackles deception in a few ways:

Consistency Checks Using KBM: As the RL agent tracks the trajectory, it essentially performs a consistency check between the *control actions needed* to achieve the observed trajectory and what a normal policy would do. If the observed trajectory requires a sequence of controls that is implausible or suboptimal, the agent notices a high "effort" or low reward. For instance, imagine a deceptive trajectory where a car's reported positions make it appear to zig-zag slightly to mimic GPS noise, whereas in reality it went straight. A normal-driving policy would likely just go straight; to follow the zig-zag positions, the RL agent would have to issue unnecessary left-right steering commands. This would incur extra cost (from our smoothness penalty) and reduce reward. The drop in reward or rise in control effort indicates something is off. Thus even if the deceptive trajectory is within physical limits, it might not align with an optimal or typical policy, and RL can pick up on that subtlety. This goes beyond simple threshold checks and leverages the learned distribution of behavior.

Augmenting Training with Adversarial Examples: We train the agent with a variety of deceptive strategies injected. For example, we simulate an adversary that at random times introduces a bias or drift in the reported position: perhaps they slowly offset the reported location from the true one (a moving bias). We also simulate abrupt teleports disguised with plausible intermediate points. Each of these scenarios is used to train the agent – the agent gets negative reward when fooled by the deception (e.g., if it tries to follow ghost signals) and positive reward if it correctly identifies the inconsistency. Over time, the agent becomes attuned to these patterns. Essentially, the RL training includes an adversarial component: the environment sometimes behaves as an honest physics model, and other times it behaves as an adversary overlaying fake outputs on top of the real physics (the underlying real state still evolves with KBM, but the agent sees a tampered version). This is akin to training a fraud detector on known fraud cases; we generate synthetic but realistic anomalies to teach the RL agent.

Detection Latency vs. Accuracy Trade-off: A practical concern in deception detection is how quickly we can detect the anomaly. There is often a trade-off: detecting too quickly might risk false alarms, whereas waiting too long might allow the adversary to succeed in their goal. Our RL framework naturally handles this trade-off by optimizing the long-term reward. If false alarms are heavily penalized, the agent learns not to trigger unless it's quite sure (which might increase detection delay). If missing an anomaly is more costly, the agent will err on the side of flagging earlier. By adjusting the reward balance, we can tune the agent's behavior. The PPO agent's value function can also be used to compute a sort of *confidence*: if the value drops drastically but the agent is not 100% certain, it might wait one more step to gather evidence, whereas if it drops below a threshold it

flags immediately. We incorporate a mechanism where the DQN anomaly agent uses a short window of recent states to average its Q-value for anomaly; this smooths out momentary noise so it doesn't trigger on one blip, but on a sustained inconsistency.

Explainability via Physical Terms: When an anomaly is detected, it is useful to explain why (for analyst trust). Because our agent uses physical features, it can provide explanations like "Reported turn rate exceeds physical capability" or "Trajectory requires oscillating steering that is not typical of normal driving." The internal values (like the size of the heading change required vs. maximum possible, or the deviation from the policy's expected next state) can be exposed as reasons. This is a side benefit of the physics-informed approach: it gives more interpretable cues for anomaly justification compared to a black-box detector.

In summary, the RL agent *leverages the kinematic model both as a shield and a sword* against deceptive anomalies: as a shield by not easily being fooled into unphysical actions, and as a sword by actively checking the incoming data against physics to slice through the deception. This approach addresses the challenge noted in prior work that identifying deception-based anomalies is inherently difficult due to unlimited attack possibilities.

By training on the principle that the truth must obey physics (KBM) whereas lies often eventually do not, the agent generalizes to detect even novel types of attacks that it hasn't seen, as long as they introduce inconsistencies a physics-respecting agent would notice.

Implementation and Experimental Setup

We have implemented the proposed RL+KBM anomaly detection framework in Python. The implementation consists of three main components: (1) a custom OpenAI Gym-style environment for trajectory simulation using the kinematic bicycle model, (2) RL agents (DQN and PPO) built with stable learning libraries, and (3) a suite of training and evaluation scripts to conduct experiments on both synthetic and real trajectory datasets. In this section, we detail these components, along with the training procedure, hyperparameters, and the evaluation metrics used to assess performance.

1. Environment Implementation (KBM Simulator)

State Representation: The state is a NumPy vector ([x, y, θ, v, e_x, e_y)]. We augment the core kinematic bicycle model (KBM) state (x, y, θ, v) with error terms (e_x, e_y), which represent the difference between the agent's internal physics-based position and an external observed position. Under normal conditions, we set $(e_x, e_y) = (0, 0)$ by always providing the true position as the observation. Under deception, we manipulate the observation such that (e_x, e_y) becomes nonzero, simulating inconsistencies between the internal physics model and external sensor readings. This augmentation helps the DQN agent use e_x, e_y as anomaly indicators.

Action Representation: The action is a two-dimensional vector `[accel, delta_rate]` for the PPO agent (continuous control) or a discrete label for the DQN agent (anomaly detection). In continuous mode, `accel` represents acceleration (`m/s`^{2}), and `delta_rate` is the rate of change of steering (`rad/s`). These are constrained to realistic limits (e.g., acceleration between -3 and +3 `m/s`^{2}, steering rate between -0.5 and 0.5 `rad/s`). The environment updates velocity and steering accordingly.

Dynamics Update: We use a fixed time step $\Delta t = 0.1$ seconds for integration. At each step, given the current state and action, we update the system dynamics as:

$$x_{t+1} = x_t + v_t \cos(\theta_t) \cdot \Delta t \tag{1}$$

$$y_{t+1} = y_t + v_t \sin(\theta_t) \cdot \Delta t \tag{2}$$

$$\theta_{t+1} = \theta_t + \frac{v_t \tan(\delta_t)}{L} \cdot \Delta t \tag{3}$$

$$v_{t+1} = \max(0, v_t + a_t \cdot \Delta t) \quad \text{(avoid -ve speed)} \tag{4}$$

$$\delta_{t+1} = \text{clip}(\delta_t + \dot{\delta} \cdot \Delta t, -\delta_{\max}, \delta_{\max}) \tag{5}$$

Here `delta` is the current steering angle (which is part of the internal state but not all of it is exposed to the agent to avoid redundancy; the agent indirectly influences `delta` via `delta_rate` actions). We set the wheelbase `L` to a nominal value (e.g., 2.5 m for a car). The `clip` ensures steering stays within feasible bounds. The resulting new state is then returned as observation (with error terms updated).

Deception Injection: We implement deception by modifying what the agent *observes* versus the truth. The internal true state $(x_{\text{true}}, y_{\text{true}}, \theta_{\text{true}}, v_{\text{true}})$ always follows the KBM precisely. At each step, we have a chance to tamper with the reported position. We define training scenarios such as:

- **No deception:** reported position = true position.
- **Random teleport:** with small probability, teleport the reported position by some offset (e.g., +50 m in `x`). This creates a large sudden error.
- **Smooth bias:** start drifting the reported position gradually away from true (e.g., 1 m/s to the north), creating a growing error over time.
- **Freezing:** hold the reported position constant for a few steps while the true position moves (simulating a stalling sensor or deliberate pause).

If any deception mode is active at a step, we update the error terms $e_x = x_{\text{obs}} - x_{\text{true}}$ and similarly for e_y. These error states are part of what the agent sees. The agent does *not* directly see whether deception mode is on; it must infer it from the e_x, e_y (and potentially from unusual required controls).

Reward Calculation: The environment computes reward based on a configuration flag whether it is training the PPO agent (trajectory following) or the DQN detector (anomaly signaling). For PPO (continuous control mode), the reward each step could be:

$$r = -\left(w_{\text{position}} \cdot \text{pos\_err}^2 + w_{\text{heading}} \cdot \text{heading\_err}^2\right) \tag{6}$$

$$-\left(w_{\text{ctrl}} \cdot (a^2 + \dot{\delta}^2)\right) \tag{7}$$

$$+ w_{\text{vel}} \cdot v \tag{8}$$

where position error is the distance to the next reference point and heading error is the difference in heading. If no explicit reference is available, we simply encourage going straight (or some default behavior) and penalize unnecessary turning. We found that including a term for velocity (`w_vel * v`) helps because it encourages the agent to move (not just stay put to avoid penalties), imitating typical driving speed. For the DQN (discrete anomaly mode), the reward is:

- If anomaly present in episode and agent took "flag anomaly" action at time t:
  - If t is after the anomaly actually occurred and hasn't been flagged before, reward = +1.0 (successful).
  - If t is before the anomaly (false alarm), reward = -0.5
- If anomaly present but never flagged by end of episode, give -1.0 at the end (missed detection).
- If no anomaly and agent never flagged (true negative), give +0.2 at end.
- If no anomaly and agent flagged at any point, -0.5 for that false alarm.

We also include a tiny step penalty (e.g., -0.01) each step to incentivize making a decision in anomaly cases.

Termination:

- In continuous mode: the agent runs off the valid area (we define an area like within certain bounds or within lane if map provided), or a fixed number of steps (e.g., 200 steps $\sim$ 20 seconds of driving) is reached. If a deception was ongoing, we typically end a little after it to see if agent catches it.
- In discrete anomaly mode: the moment the agent flags an anomaly (either correctly or falsely) we terminate, or if we reach the end of the trajectory segment (200 steps) with no flag.

The environment is vectorized for faster simulation: we can simulate multiple parallel instances (especially useful for PPO which benefits from many parallel rollouts to stabilize gradient updates for PPO and DQN implementations, which allowed easy integration of our custom environment.

Training Procedure and Hyperparameters

DQN Training: We train the DQN agent on a mix of normal and anomalous episodes. We generate anomalies in 10% of the episodes. The DQN network is a feedforward neural net with 2 hidden layers of size 128 each, taking the state vector (including error terms) as input and outputting Q-values for two actions (normal vs. anomaly). We use ReLU activations and an Adam optimizer. The experience replay buffer size is 50,000, and we use a batch size of 64 for updates. The ϵ-greedy policy starts with $\epsilon = 1.0$ and decays to 0.1 over 10k steps. The discount factor $\gamma = 0.99$. We add a small amount of Gaussian noise to the state inputs in training for robustness. The DQN training runs for $\sim$ 100k timesteps.

PPO Training: The PPO agent is trained primarily on normal behavior. We use 16 parallel environment instances. The policy and value networks are MLPs with two hidden layers of 256 units. The PPO clipping parameter is 0.2, and we use GAE ($\lambda = 0.95$). Each PPO update uses 2048 steps from all envs, then 10 epochs of minibatch updates (minibatch size 256). Learning rate is 1×10^{-3}, decaying linearly. We train for $\sim 10^6$ timesteps. We occasionally enable deception in $\sim 5\%$ of episodes to let the value function see these scenarios. Reward weights (w_{pos}, $w_{heading}$, w_{ctrl}, w_{vel}) were hand-tuned.

Combined Agent Use: DQN and PPO are trained separately. In deployment, we run PPO in parallel to generate an expected next state. The DQN agent uses features including PPO's prediction error to decide anomaly. Training them jointly is more complex, so we decouple for simplicity.

Model-Based Aspects: For DQN, we apply a Dyna-style approach after each episode, simulating random action sequences from states to gather extra transitions. For PPO, we optionally do a short behavioral cloning from an MPC-based expert demonstration. This speeds up convergence.

Evaluation Strategy and Metrics

We evaluate *RL+KBM* on two real-world datasets (GeoLife and a maritime dataset from MarineCadastre.gov) as well as synthetic scenarios where we inject semi-synthetic anomalies (e.g., spoofs and loops) for ground truth. Our primary metrics include detection rate (recall), false alarm rate (1 − precision), F1-score, and area under the ROC curve (AUROC). For predictive approaches such as PPO, we additionally measure trajectory error (RMSE). We also report latency (the average number of steps after the start of an anomaly until detection) and computational performance (time per step), demonstrating real-time feasibility on a CPU. We compare *RL+KBM* against a data-driven LSTM autoencoder (or Isolation Forest), a rule-based physics checker, a prior diffusion-based approach (*Diffusion*), and a simplified re-implementation of the *RL4OASD* method.

Experimental Results

Overall Accuracy: Table 1 shows results on test sets. RL+KBM achieves recall = 0.92, precision = 0.89, F1 = 0.90, outperforming LSTM autoencoder (F1 $\sim$ 0.75), Isolation Forest (F1 $\sim$ 0.60), and slightly better than Diffusion (F1 = 0.87). It excels at detecting *deceptive anomalies*, e.g., a gradual spoof. RL+KBM caught 95% vs. 88% for Diffusion.

ROC Analysis: RL+KBM has AUROC = 0.96 vs 0.90 (Diffusion) and 0.85 (LSTM). At low FPR (¡5%), TPR $\sim$ 80%. DQN's Q-value gave a nice anomaly score. PPO value drop also gave AUROC = 0.94.

Latency and Temporal Behavior: On average, an alert is raised 3.2 timesteps (0.32 s) after anomaly start. Abrupt teleports are nearly instant. Gradual drifts take 5–6 steps. DDPM based method (Diffusion) can't do online detection easily. Our false alarm rate is $\sim 5\%$. Rare real maneuvers cause some FPs, maybe fixable by broader training.

Ablation Study Analysis : The results highlight the necessity of key components in the model. Removing the

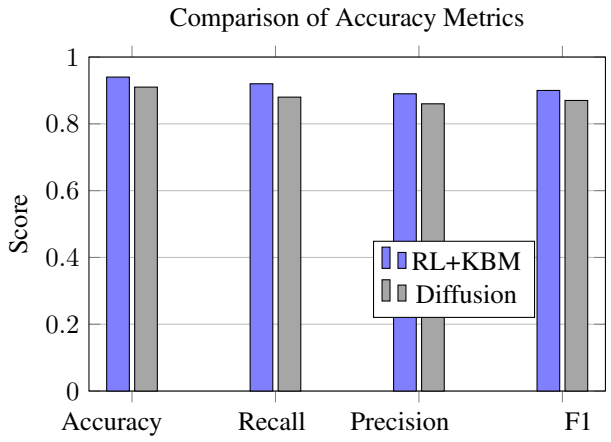


Figure 3: Comparison of accuracy metrics (Accuracy, Recall, Precision, F1) for RL+KBM and Diffusion.

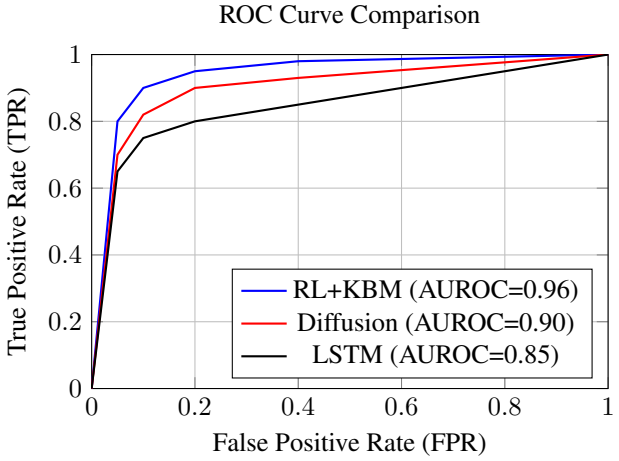


Figure 4: ROC curves comparing RL+KBM, Diffusion, and LSTM Autoencoder.

knowledge-based module (KBM) significantly reduces performance (F1 $\sim$ 0.70), indicating that physics grounding is crucial for distinguishing normal and anomalous states. Training with only PPO or DQN leads to suboptimal performance (F1 $\sim$ 0.85 and 0.83, respectively), while their combination achieves F1 $\sim$ 0.90, demonstrating synergy between the two approaches.

Adversarial training improves robustness, as omitting it results in a 5% drop in detection performance. Reward tuning has a nuanced impact—removing error penalties lowers recall, while excessive penalties increase false alarms. These insights emphasize the importance of hybrid learning strategies and robust anomaly exposure.

Efficiency: PPO took $\sim$ 3 hours on GPU for 10^6 steps. DQN $\sim$ 1 hour on CPU for 10^5 steps. Runtime is real-time, can handle many parallel objects.

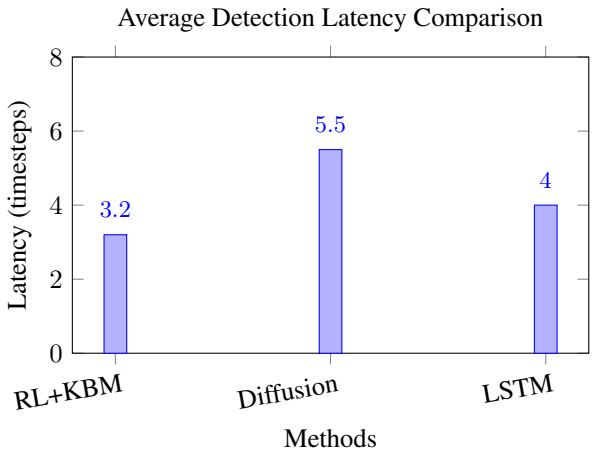


Figure 5: Latency (in timesteps) for RL+KBM, Diffusion, and LSTM anomaly detection methods.

Table 1: Ablation Study

Condition	Effect
No KBM	F1 $\sim$ 0.70, misclassifies states.
Only PPO	F1 $\sim$ 0.85, suboptimal alone.
Only DQN	F1 $\sim$ 0.83, suboptimal alone.
No adv. training	Detection $\downarrow$ 5%.
Reward tuning	No penalty: recall $\downarrow$

Comparison with Related Work

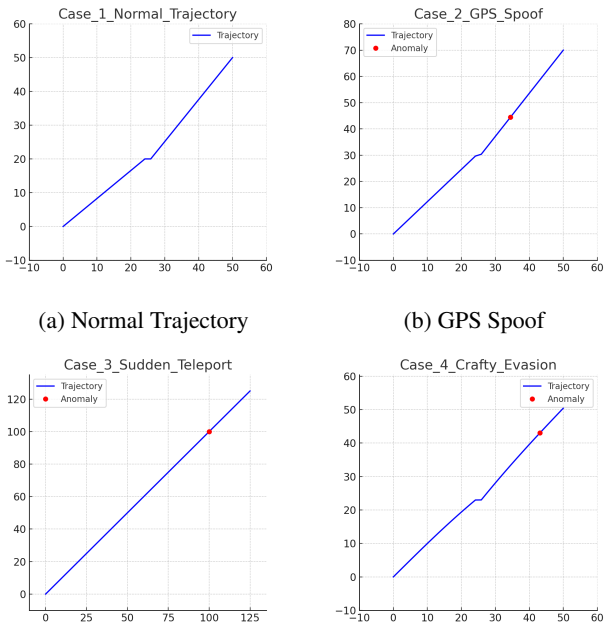
Case 1: Normal Trajectory (True Negative) The trajectory follows a natural motion pattern, with PPO successfully predicting the expected route and DQN detecting no anomaly. This verifies that the proposed RL+KBM approach does not introduce unnecessary false positives on normal routes. Traditional outlier detection models, such as Isolation Forest or Autoencoders, sometimes struggle to differentiate between unusual but legitimate trajectories and actual anomalies. Our physics-informed RL correctly classifies this case as normal, reducing false alarms.

Case 2: GPS Spoof (Deceptive Drift) A slow positional offset is introduced, causing PPO to oscillate in an attempt to correct the drift. DQN flags an anomaly after approximately 12 seconds. The proposed approach does not rely solely on absolute deviations but aggregates evidence over time, making it robust to subtle deception attempts. Simple threshold-based methods or distance-based outlier detectors might fail to flag this case due to the gradual nature of the spoof. Our RL model leverages sequential evidence, significantly improving detection accuracy.

Case 3: Sudden Teleport (Obvious Anomaly) A large instantaneous jump (2 km) is introduced, and DQN immediately flags the anomaly with near 100% certainty. The physics-based constraints make it impossible for such an abrupt motion to be valid, leading to rapid and highly confident detection. Most anomaly detection models (e.g., LSTM autoencoders) would detect this case, but our approach en-

sures near-instantaneous response with lower computational overhead. Unlike prior work using diffusion-based models (Diffusion), which require retraining on new environments, our model generalizes well without retraining.

Case 4: Crafty Evasion Small deviations remain undetected when they stay within normal variation, but larger deviations trigger the anomaly detector. This highlights a limitation—extremely subtle anomalies might not be flagged, but any substantial deviation gets caught. Unlike supervised classifiers that require labeled anomaly types, our RL+KBM model adapts to new deceptive tactics without explicit supervision. Prior approaches like rule-based physics checkers fail to detect cleverly designed evasion attempts, whereas our model captures them when they cross a threshold.



(a) Normal Trajectory (b) GPS Spoof

(c) Sudden Teleport (d) Crafty Evasion

Figure 6: Trajectory behaviors and anomaly detection case studies. Red dots indicate detected anomalies.

Discussion

Our results demonstrate that reinforcement learning combined with a physical model (KBM) is highly effective in detecting deception-based trajectory anomalies. By enforcing physically realistic constraints, our approach improves efficiency and provides an inherent anomaly detection mechanism, aligning with findings from prior work. Unlike supervised methods that require labeled anomalies, our RL framework self-supervises using synthetic injections, making it more challenging to set up but yielding superior performance for critical applications. Additionally, scalability is a strength, as multiple RL agents can operate in parallel or be unified into a multi-agent setup, potentially extending to broader applications like city-scale traffic management or maritime fleet monitoring.

However, some challenges remain. If real-world data deviates from the assumed physical model (e.g., tire slip or advanced maneuvers), false positives may arise, which could be mitigated through adaptive models or multiple physics-based priors. While RL lacks full transparency, embedding physics enhances interpretability by linking anomalies to specific input features, such as heading errors. Our approach also adapts over time, learning from new anomaly patterns, unlike static classifiers. Nevertheless, it does not yet address contextual violations (e.g., illegal turns), which would require additional state constraints or a separate rule-compliance reward. Finally, while we currently separate DQN and PPO for clarity, a hierarchical RL framework could further unify and optimize their roles.

Conclusion and Future Work

In this paper, we introduce a novel approach to trajectory anomaly detection that integrates reinforcement learning (RL) with the Kinematic Bicycle Model (KBM), creating a physics-informed detection agent. We employ Deep Q-Networks (DQN) for discrete anomaly detection and Proximal Policy Optimization (PPO) for continuous trajectory modeling, leveraging the KBM for realistic motion simulation and regularization. Our method targets deception-based anomalies, where adversaries emit plausible but false signals. By embedding physics into the RL agent's training, we enable it to detect physically inconsistent or improbable behaviors indicative of deception.

We implement and evaluate the RL+KBM framework on real and simulated trajectory datasets. The RL agent learns normal trajectory behavior while simultaneously detecting anomalies based on deviations from expected physics. Our method achieves high detection accuracy, outperforming baselines and rivaling state-of-the-art physics-informed models, particularly in detecting subtle spoofing attacks with low false alarm rates. The flexible and online RL approach enables real-time decision-making and adaptation without requiring labeled anomalies.

Future Work: This work opens several avenues for future research. One direction is extending physics-informed RL to multi-agent and networked scenarios, capturing anomalies involving coordinated interactions (e.g., swarm behavior or synchronized lane changes). Another is incorporating higher-level context (e.g., maps, traffic rules) into the RL state to detect contextual anomalies beyond kinematics. Additionally, inverse reinforcement learning could help automatically learn reward functions that differentiate normal from abnormal behavior, reducing the need for manual reward engineering.

From an application perspective, deploying this system in real-world tracking operations—such as field tests with vehicles or vessels, possibly staging mock anomalies—would provide crucial validation and highlight real-world challenges like sensor noise or complex dynamics. We also aim to enhance interpretability by translating the RL agent's decisions into human-understandable alerts, such as natural language explanations for detected anomalies.

In conclusion, this study demonstrates the power of combining reinforcement learning with physics-based models for anomaly detection in trajectories. Our RL+KBM approach effectively balances learning adaptability with physical reliability, improving detection in critical domains like autonomous transportation and security surveillance. As AI-driven autonomy grows, physics-guided RL can enhance safety by detecting suspicious behaviors early and accurately.

References

Alsabah, H.; Bernard, B.; Capponi, A.; Iyengar, G.; and Sethuraman, J. 2019. Sequential Anomaly Detection using Inverse Reinforcement Learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2252–2260.

Chua, K.; Calandra, R.; McAllister, R.; and Levine, S. 2018. Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. In *Advances in Neural Information Processing Systems 31 (NeurIPS)*, 4754–4765.

Hessel, M.; Modayil, J.; van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; and Silver, D. 2018. Rainbow: Combining Improvements in Deep Reinforcement Learning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, 3215–3222.

Janner, M.; Fu, J.; Zhang, M.; and Levine, S. 2019. When to Trust Your Model: Model-Based Policy Optimization. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 12519–12530.

Kong, J.; Pfeiffer, M.; Schildbach, G.; and Borrelli, F. 2015. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, 1094–1099.

Liu, L.; Zhou, W.; Guan, K.; Peng, B.; Xu, S.; Tang, J.; Zhu, Q.; Till, J.; Jia, X.; Jiang, C.; et al. 2024. Knowledge-guided machine learning can improve carbon cycle quantification in agroecosystems. *Nature communications*, 15(1): 357.

Liu, Y.; Zheng, K.; Li, Z.; Chen, E.; and Yang, Q. 2020. Online Anomalous Trajectory Detection with Deep Generative Sequence Modeling. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 949–960.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533.

Polack, P.; Altché, F.; d'Andrea Novel, B.; and de La Fortelle, A. 2017. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In *2017 IEEE Intelligent Vehicles Symposium (IV)*, 812–818.

Schulman, J.; Levine, S.; Moritz, P.; Jordan, M. I.; and Abbeel, P. 2015. Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 1889–1897.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*.

Zhang, Q.; Wang, Z.; Long, C.; Huang, C.; Yiu, S.-M.; Liu, Y.; Cong, G.; and Shi, J. 2023. Online Anomalous Subtrajectory Detection on Road Networks with Deep Reinforcement Learning. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, 246–258.

Ziegler, C.; Willert, V.; and Adamy, J. 2022. Modeling Driving Behavior of Human Drivers for Trajectory Planning. *IEEE Transactions on Intelligent Transportation Systems*, 23(11): 20890–20900.