Trust Scoring of Industrial IoT Devices

Abhijeet Antony Tomy
Advisor : Dr. Pushpak Jagtap

***Abstract*— The success of IoT deployment largely depends on applications, privacy, security, and trust. Trust is vital in developing trustworthiness and connectivity across devices to provide secure services. International Data Corporation (IDC) forecasts that there will be 41.6 billion IoT devices in 2025 capable of generating 79.4 zettabytes (ZB) of data; therefore, each device in the IoT system must be trustworthy. In this project, we are proposing a methodology to measure the trust of Industrial IoT Devices using the Key trust indicators (KTI) from the device.**

I. INTRODUCTION

Trust and security, although often used interchangeably, are distinct concepts. While a system may be secure, meaning it has implemented protective measures, if these measures can be bypassed without detection, the system cannot be considered trustworthy. Similarly, a system may be trusted, implying that it behaves as expected and can identify and communicate issues, but if it harbours undiscovered software vulnerabilities, it is not truly secure. Trust involves expected behaviour and the ability to recognize and communicate problems, whereas security involves effective protective measures against potential threats.

Trustworthiness is the degree to which a system satisfies its owners' and users' expectations. Thus, trustworthiness is system and mission-specific.

Physical damage, weather conditions, and component maintenance impose uncertainty that is inherent in embedded systems but seldom seen in general-purpose computing systems. In the physical world of embedded systems, all operational environments are adversarial, whether by accident or intent, whether by users, developers, or others, and whether by intelligent or natural adversaries or some combination of these.

Given this backdrop, the imperative emerges for a metric to measure trust, one that not only provides a quantitative assessment but also emphasizes the transparency of the assigned score. In this context, an explainable metric becomes paramount to effectively gauge and communicate embedded systems' trustworthiness. This paper delves into the development of such a metric, considering the nuanced interplay between security, trust, and the specific challenges posed by the adversarial nature of embedded system environments.

We start by examining the necessary mathematical concepts and tools in Section II. Following that, in Section III, we establish the problem formulation. We formally outline the requirements, emphasizing the application, and proceed to set up the IoT test bed for data generation in Section IV-A. Additionally, we design the trust scoring algorithm in Section IV-B

II. PRELIMINARIES

A. Reinforcement Learning

Reinforcement Learning (RL) involves learning an optimal policy in the context of a Markov Decision Process (MDP)[1] defined by the tuple $(S, A, \mathcal{T}, \mathcal{T}_0, \mathcal{R}, H)$, where:

S is the set of states representing possible situations,
A is the set of actions representing the possible decisions,
$\mathcal{T}$ is the transition function describing how the environment evolves give
$\mathcal{T}_0$ is the initial state distribution,
$\mathcal{R}$ is the reward function indicating the immediate feedback for an actio
H is the time horizon.

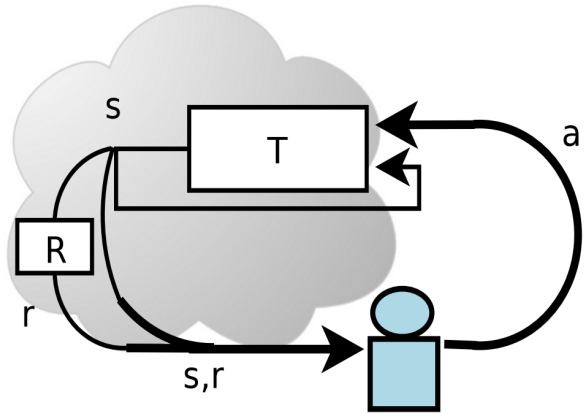

Fig. 1. A schematic showing the subject agent (shaded in blue) performing RL [2]. In forward learning or RL, the agent chooses an action at a known state and receives a reward in return generated by a reward function R that may not be known to the agent. The state changes based on the previous state and action, which is modelled using the transition function T that may be unknown as well.

At each time step t, the agent observes the current state $s_t \in S$, selects an action $a_t \in A$ based on its policy π, receives a reward $r_t = \mathcal{R}(s_t, a_t)$, and transitions to the next state $s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)$.

The objective is to learn a policy π that maximizes the expected cumulative reward:

$$J(\pi) = E_{\tau\sim\pi}\left[\sum_{t=0}^{H-1} \mathcal{R}(s_t, a_t, s_{t+1})\right]$$

where $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{H-1}, a_{H-1}, r_{H-1}, s_H)$ is a trajectory, and the expectation is over the stochasticity present in the transition function, reward function, and policy.

The policy π maps a state to a distribution over the action space: $a_t \sim \pi(\cdot|s_t)$. A fundamental concept is the state-value function $V^\pi(s)$, representing the expected cumulative reward from state s following policy π:

$$V^\pi(s) = E_{\tau\sim\pi}\left[\sum_{t=0}^{H-1} \mathcal{R}(s_t, a_t, s_{t+1}) \mid s_0 = s\right]$$

In Deep Reinforcement Learning[3], the policy π is often parameterized by a deep neural network. Various algorithms, such as Q-learning[4] and policy gradient methods[5], aim to optimize the parameters of this network to improve the agent's performance.

While RL provides a versatile framework, challenges include exploration strategies, addressing the exploration-exploitation dilemma, and dealing with high-dimensional state and action spaces. Deep Reinforcement Learning techniques address these challenges by leveraging the representational power of neural networks to handle complex input spaces.

B. Invrse Reinforcement Learning

Inverse reinforcement learning (IRL) is the problem of modelling the preferences of another agent using its observed behaviour, thereby avoiding a manual specification of its reward function [6] [7]. Approaches for IRL predominantly ascribe a Markov decision process (MDP) [1] to the interaction of the observed agent with its environment and whose solution is a policy that maps states to actions. The reward function of this MDP is unknown, and the observed agent is assumed to follow an optimal policy of the MDP. Typically, if a designer wants intelligent behaviour in an agent, she manually formulates the problem as a forward learning or forward control task solvable using solution techniques in RL, optimal control, or predictive control. A key element of this formulation is a specification of the agent's preferences and goals via a reward function. The need to pre-specify the reward function limits the applicability of RL and optimal control to problems where a reward function can be easily specified. IRL offers a way to broaden the applicability of RL and reduce the manual design of task specification, given a policy or demonstration of desired behaviour is available. While acquiring the complete desired policy is usually infeasible, we have easier access to demonstrations of behaviours, often in the form of recorded data. Thus, IRL forms a key method for learning from demonstration [8]. IRL is formally defined as: Let an MDP without reward, M_{IRL}, model the interaction of the expert E with the environment. Let $D = \{\tau_i\}_{i=1}^N$, where each τ_i is a demonstrated trajectory $\tau_i = (s_0, a_0, s_1, a_1, \ldots, s_j, a_j)$. Here, $s_j \in S$, $a_j \in A$, and $i, j, N \in N$ represent the set of demonstrated trajectories. A trajectory in D is denoted as τ. We may assume that all $\tau \in D$ are perfectly observed. Then, determine $\hat{R}_E$ that best explains either policy π_E if given or the observed behaviour in the form of demonstrated trajectories. Notice that Inverse Reinforcement Learning (IRL) inverts the Reinforcement Learning (RL) problem. Whereas RL seeks to learn the optimal behaviour based on experiences (s, a, r) or (s, a, r, s') that include obtained rewards, IRL seeks to best explain the observed behaviour by learning the corresponding reward function.

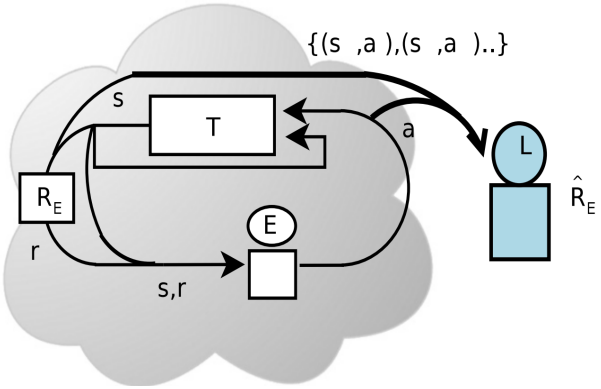

Fig. 2. In Inverse Reinforcement Learning (IRL), the input and output for the learner L are reversed. L perceives the states and actions $\{(s, a), (s, a), \ldots, (s, a)\}$ of expert E (or its policy π_E), and learns a reward function $\hat{R}_E$ that best explains E's behavior as the output. Note that the learned reward function may not exactly correspond to the true reward function.

III. Problem Formulation

In the initial phase, the device is operated under optimal conditions, with the assumption that it will function in accordance with specifications. A vigilant observation is conducted throughout this operational period to discern the system's behaviour, and a comprehensive set of Key Trust Indicators (KTIs) is systematically gathered. Within the context of our experiment's design, the criteria for ideal conditions are outlined as follows:

- The communication channel between the Raspberry Pi and ESP8266 remains consistently intact.
- The router facilitating wireless communication between the Raspberry Pi and ESP8266 is devoid of any external loads.
- Solely the PID loop, responsible for generating motor control commands, is operational on the Raspberry Pi.
- Each hardware component operates without any faults.

Under the aforementioned set of ideal conditions, an exhaustive collection of KTI's is amassed to intricately formulate a policy that encapsulates the essence of ideal behaviour. Once this policy is successfully discerned, the experimentation proceeds to a subsequent phase, introducing a spectrum of diverse attacks into the system. These attacks encompass a multifaceted array, including:

- The deliberate transmission of malicious motor control commands from an untrusted Raspberry Pi.
- Disruption of the communication channel by strategically turning off wireless communication.
- Execution of a process on the Raspberry Pi that conspicuously consumes substantial CPU and memory resources.
- Augmentation of the load on the router facilitating communication between ESP8266 and Raspberry Pi.

- Deliberate failure to dispatch motor control commands from the Raspberry Pi.
- Intentional omission of the transmission of encoder information from ESP8266 to Raspberry Pi.
- Systematically introducing hardware failures into the operational environment.

In the subsequent phase, the system is observed under these adversarial conditions, and a collection of KTI's is undertaken. To adapt and comprehend the intricacies of the system's behaviour under adversarial circumstances, the experimental framework leverages the sophisticated methodology of Inverse Reinforcement Learning (IRL) to derive a nuanced reward function tailored specifically for these complex scenarios.

IV. Proposed Methodology

The proposed solution bifurcates the problem into two distinct yet interrelated components: 1) Setup IoT test bed - to collect Key trust indicators and 2) Design device trust scoring algorithm. This methodology is designed so that the data required for the algorithm can be generated by the test bed.

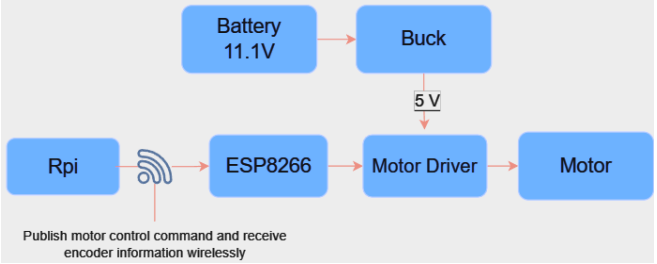

Fig. 3. Block diagram of the IoT test bed

A. Setup IoT test bed

The IoT testbed uses a simple client-server architecture to establish bidirectional communication between an ESP8266 microcontroller and a Raspberry Pi. The ESP8266, a widely used Wi-Fi module, is programmed to receive messages from the Raspberry Pi, perform specific actions, control the motor upon message reception, and then transmit the current encoder information from the motor back to Raspberry Pi. The primary goal is to run a PID controller to control the motor wirelessly between the ESP8266 and a Raspberry Pi, showcasing how IoT devices can seamlessly interact with computing systems, The block diagram for the IoT test bed is shown in figure 3.

1) Hardware Setup: The hardware setup comprises a Raspberry Pi, an ESP8266, a DC motor, and a motor driver (MD10c R3). A battery is connected to the motor driver, linked to the ESP8266's ground. The PWM pin of the motor driver connects to D5 (Pin 14) of the ESP8266, while the DIR pin connects to D6 (Pin 12). The motor driver interfaces with the DC motor, and the motor's ground connects to the ESP8266's ground—the motor's VCC links to the ESP8266's Vin. The directional control pins (M+ and M-) are connected to D7 (Pin 13) and D8 (Pin 15) of the ESP8266, respectively. This configuration enables the Raspberry Pi to wirelessly control the motor through the ESP8266, with PWM commands regulating the motor speed. The connection for the hardware setup is shown in figure 4

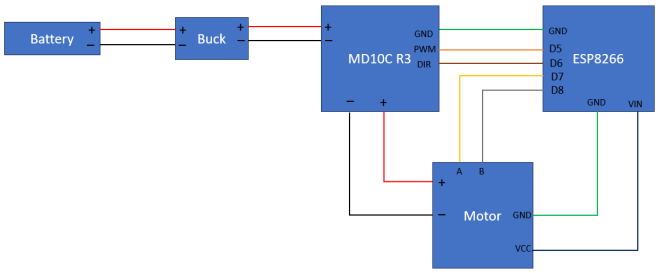

Fig. 4. Pin diagram

2) Message Flow: The message flow starts with the Raspberry Pi sending a control input to the ESP8266. Upon receiving the message, the microcontroller sets the motor speed according to the received message. Subsequently, the ESP8266 sends the encoder information back to the Raspberry Pi where PID control runs. This bidirectional exchange demonstrates the reliability and responsiveness of the communication protocol.

3) Collecting KTI: A logger is activated upon establishing bidirectional communication between the ESP8266 and the Raspberry Pi for message flow. This logger records various parameters, including the PWM message transmitted, CPU usage, memory usage, network activity of the Raspberry Pi, and the message received from the ESP8266.

B. Design device trust scoring algorithm

The test bed serves as the operational environment for subjecting the device to ideal conditions, facilitating the collection of Key Trust Indicators (KTIs) to derive a policy using reinforcement learning that effectively encapsulates the ideal behaviour exhibited by the system. Transitioning to the subsequent phase, we deliberately introduce various attacks to simulate adversarial conditions, concurrently gathering KTI data for the system. Leveraging Inverse Reinforcement Learning (IRL), we extract the reward trajectory associated with these adversarial scenarios. The disparity between the initially established ideal reward function and the subsequently learned reward function constitutes the trust score attributed to the device.

V. Work done and Intermediary Results

A bidirectional communication test bed for the Internet of Things (IoT) was effectively established between Raspberry Pi and ESP8266. In each cycle, the ESP8266 transmits motor velocity, which is then received by the Raspberry Pi. Subsequently, the Raspberry Pi calculates proportional, integral, and differential gains for PID control and returns the resulting value to the ESP8266. Throughout each iteration, various metrics such as CPU usage, memory usage, received velocity, and the transmitted control input (u) are monitored

and recorded in a CSV file. A snapshot of this data is shown in figure 5

Iteration	Received Velocity	PID contrc	CPU Usag	Memory Usage	Sent U
1	0.69	307.1175	0.8	49.1	307.1175
2	0.12	339.1334	0.2	48.8	339.1334
3	28.84	268.464	0.3	48.6	268.464
4	29.19	268.4723	0.2	48.6	268.4723
5	29.22	269.2029	0.3	48.6	269.2029
6	29.25	269.8914	0.2	48.6	269.8914
7	29.25	270.6594	0.3	48.6	270.6594
8	29.16	271.7414	0.7	48.6	271.7414
9	29.09	272.8482	0.8	48.7	272.8482
10	28.93	274.3483	0.7	48.8	274.3483
11	28.88	275.6178	0.6	48.7	275.6178
12	28.86	276.8373	0.4	48.5	276.8373
13	28.88	277.9252	0.6	48.6	277.9252
14	28.96	278.7963	0.5	48.6	278.7963
15	28.9	280.0751	0.6	48.6	280.0751
16	28.79	281.5869	0.5	48.5	281.5869
17	28.85	282.6302	0.4	48.4	282.6302
18	28.82	283.9066	1.1	48.6	283.9066
19	28.95	284.6607	0.8	48.5	284.6607
20	29.06	285.3464	0.5	48.3	285.3464

Fig. 5. A snapshot of information recorded when the device was being operated in ideal condition

VI. Future Work

Utilizing data gathered under optimal circumstances, acquiring a policy through Reinforcement Learning is imperative. Subsequently, the system's behaviour must be observed under various attacks, and Inverse Reinforcement Learning (IRL) should be employed to obtain the current system function. The trust score will be determined by evaluating the disparity between the function learned under ideal conditions and the function learned after the introduction of attacks.

Acknowledgments

This project is funded by Siemens and we would like to thank Dr.Kapaleeswaran Viswanathan from the cyber security research team for his continued support.

References

[1] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[2] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[3] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau, *et al.*, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[4] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[5] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[6] S. Russell, "Learning agents for uncertain environments," in *Proceedings of the eleventh annual conference on Computational learning theory*, pp. 101–103, 1998.

[7] A. Y. Ng, S. Russell, *et al.*, "Algorithms for inverse reinforcement learning.," in *Icml*, vol. 1, p. 2, 2000.

[8] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.