Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 207 (2022) 3656–3665

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Knowledge Based and Intelligent Information and Engineering Systems, KES2022, 7-9 September 2022, Verona, Italy

RAMARL: Robustness Analysis with Multi-Agent Reinforcement Learning - Robust Reasoning in Autonomous Cyber-Physical Systems

Aya Saad[a,b,*], Anne Håkansson[a,c]

[a]*Department of Engineering Cybernetics, The Norwegian University of Science and Technology (NTNU), Trondheim, Norway*
[b]*Aquaculture Department, SINTEF Ocean AS, Trondheim, Norway*
[c]*Department of Computer Science, The Arctic University of Norway (UiT), Tromsø Norway*

Abstract

A key driver to offering smart services is an infrastructure of Cyber-Physical systems (CPS)s. By definition, CPSs are intertwined physical and computational components that integrate physical behaviour with computation. The reason is to autonomously execute a task or a set of tasks providing a service or a list of end-users services. In real-life applications, CPSs operate in dynamically changing surroundings characterized by unexpected or unpredictable situations. Such operations involve complex interactions between multiple intelligent agents in a highly non-stationary environment. For safety reasons, a CPS should withstand a certain amount of disruption and exert the operations in a stable and robust manner when performing complex tasks. Recent advances in reinforcement learning have proven suitable for enabling multi-agents to robustly adapt to their environment, yet they often depend on a massive amount of training data and experiences. In these cases, robustness analysis outlines necessary components and specifications in a framework, ensuring reliable and stable behaviour while considering the dynamicity of the environment.

This paper presents a combination of multi-agent reinforcement learning with robustness analysis shaping a cyber-physical system infrastructure that reasons robustly in a dynamically changing environment. The combination strengthens the reinforcement learning, increasing the reliability and flexibility of the system by applying robustness analysis. Robustness analysis identifies vulnerability issues when the system interacts within a dynamically changing environment. Based on this identification, when incorporated into the system, robustness analysis suggests robust solutions and actions rather than optimal ones provided by reinforcement learning alone. Results from the combination show that this infrastructure can enable reliable operations with the flexibility to adapt to the changing environment dynamics.

Keywords: Robust reasoning; robustness analysis; autonomous cyber-physical systems (CPS); dynamic environment; Reinforcement Learning

* Corresponding author.
E-mail address: aya.saad.ntnu.no@gmail.com

10.1016/j.procs.2022.09.426

1. Introduction

Sensors and communication technologies have emerged rapidly, enabling to collect a lot of information at a very high level of detail. Such vast amount of detailed information allows sophisticated analysis to take place. Analyzing enormous information yields a better understanding of the insightful meanings behind the captured data. This further boosts the ability to better optimize the human and the community's daily life activities through the enablement of CPSs, which can act and react through cognitive reasoning processes that sometimes can occur in a distributed manner [25].

CPSs are intertwined computation, physical processes, communications, and networking components, where the computation controls the hardware in order to perform a specific task. CPSs integrate physical behaviour with computation to **autonomously** execute a task or a set of tasks to **provide a service** or a list of services for **end-users**' (i.e., individuals, group of individuals, organizations). These CPSs describe embedded systems, sensors, and control systems that are enabled by computational algorithms.

The CPS surrounding environment is likely to have a dynamic nature for real-world applications. The system needs to have an adaptable behaviour that should behave the same irrespective of the environment's dynamicity. That means a CPS should be robust and securely fulfill a service with a certain level of quality, even with the introduction of faults or when unforeseen situations in the surrounding environment occur. To this aim, a robust (resilient) CPS design should ensure the system's stability, security, and systematicity. The design process of an entire robust CPS includes concepts and modelling of control, security, and cyber-physical coupling, where the connections between security and control models should be drawn [17]. Reliability is an essential characteristic of a well-designed robust CPS. Nevertheless, it is hard to achieve robustness since the computer programs and algorithms are naturally exerted in a discrete manner. Simultaneously, the physical actions, which CPSs tend to control against the dynamically changing surrounding environment, are continuous.

Recent breakthroughs in AI and especially in reinforcement learning (RL) of multi-agent systems have realized many real-life applications. RL-based approaches enable multi-agents to adapt to the environment robustly, yet they often depend on a massive amount of training data and experiences. Ideally, CPS should work in a robust, distributed, and collaborative manner while making smart and cognitive decisions and finding better, quicker ways to solve data-driven problems. With robustness analysis, the CPSs get robust solutions to execute, rather than deterministic optimal ones as the reinforcement learning provides.

The main contribution of this work is to present a combination of reinforcement learning with robustness analysis. Utilizing reinforcement learning in multi-agent systems guarantees the choice of optimal policies. On the other hand, when the multi-CPS system is augmented by robustness analysis, it can realize robust behaviour. This combination ensures the system's flexibility and reliability.

This paper presents the different building blocks of the infrastructure **RAMARL** formed by a network of multi-agents that use reinforcement learning as their reasoning strategy. The work shows how robustness analysis can be combined with the different components of the infrastructure. The result is an infrastructure of multi-cyber-physical systems RAMARL that reason and behave robustly in a dynamically changing environment.

2. Related work

To illustrate how CPS can be integrated to form smart infrastructures, different CPS architectures and frameworks are proposed in the literature. The previously researched attempts serve a specific purpose. For instance, in [1] a review on a CPS architecture with 5 components for manufacturing based on standards is presented. The 5 components, from the hardware connection up to the application level, are listed as follows:

1. Connection for Condition Based Monitoring (CBM) sets up protocols for data transfer and realization;
2. Conversion for Prognostics and Health Management (PHM) is responsible for converting captured data into information to bring self-awareness;
3. Cyber for Cyber-Physical Systems (CPS) represents the central knowledge that includes all the information gathered and their analysis;
4. Cognition for Decision Support System (DSS) provides insights on the decision-making, the alternative actions the system would follow to achieve the goal, and the consequences of interacting with the environment;
5. Configure for Resilient Control System (RCS) is concerned with providing cyber-physical feedback.

Another work in [13] introduced a smart and volatile ICT infrastructure that is human-centric, so-called **Ipsum**. The devices and services attached to this infrastructure are ubiquitous. These devices are built on common ICT infrastructures, including sensors, software, and systems, that support operational task coordination. Ipsum's main goal is to illustrate how the different devices and services can be brought together to support smart cities and communities based on personalized services.

The work in this paper, is based on previous research presented in [14] which outlined the varying reasoning strategies and robustness analysis along with their advantages and drawbacks to enable CPS to reason robustly. As a prolongation, this paper shows how deep reinforcement learning, one of the non-classical reasoning strategies, is combined with robustness analysis to carry out robust reasoning in multi-agent autonomous CPS. The contribution is a common CPSs infrastructure handling robust reasoning in dynamic environments.

3. The RAMARL System

The RAMARL system is an infrastructure of CPSs forming an intelligent multi-agent reinforcement learning (MARL) system. Individual CPS in the system are assigned a certain goal while interacting with other systems. Individual CPS in the system is assigned a certain goal while interacting with other systems. Each CPS typically works in a closed-loop fashion. The CPS takes the data from the sensors and adapts its actions based on this sensing data to handle, for instance, distance keeping, path tracing, or object tracking.

CPSs have been used as the core component of a wide range of applications, such as applications together with the Internet of Things (IoT), autonomous systems, e.g., smart autonomous robots, unmanned autonomous air, underwater or surface vehicles, self-driving cars, and Smart X technologies as in smart grid, smart manufacturing, smart transportation, smart cities, smart health. In these smart X technologies, the CPSs can interact in a distributed manner to perform desired tasks. Smart CPSs are considered the next generation of manufacturing development in the Industry 4.0 framework with self-organization, self-diagnosis, and self-healing capabilities facilitating them to adapt to continuously changing manufacturing requirements. In manufacturing, the CPS is the core component of a service-oriented architecture equipped with cognitive capabilities such as perception, reasoning, learning, and cooperation. Setting up proper architectures and standards for information and knowledge exchange in such CPSs is still in its early stages [1, 29, 33, 4].

Ideally, autonomous CPS systems should work in a robust, distributed, and collaborative manner while making smart and cognitive decisions and finding better, quicker ways to solve data-driven problems. The proposed infrastructure follows a modular approach to facilitate the flexibility and agility of the components. CPSs are modelled to work in collaboration. They are by nature agile, potentially disposable, and can be deployed in large numbers leading to decentralized information management and decision making [25, 4].

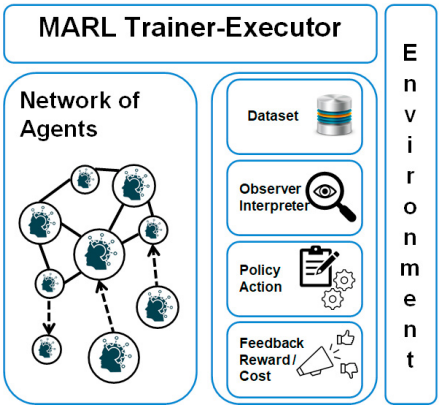

Fig. 1. RAMARL proposed system

3.1. The network of agents infrastructure

The core component of the infrastructure, as illustrated by the network of agents in fig. 1, is the controller system that defines the system agents at three levels: individual agents, multiple agents performing a single task, and the third

level is the control over the full system. Agents' beliefs, goals, and intentions are modelled through a multimodal Belief-Goal-Intension (BGI) approach [10, 15, 8, 12, 6, 5].

A multi-agent robust CPS concept is concerned with an entire system of multiple agents interacting together through negotiations and coordination in a secure manner while considering the whole system's resilience, flexibility, scalability, and adaptability to environmental changes. Mission-based intelligent multi-agents exert situation awareness, analyze and infer information from the knowledge base, and determine precise interventions to achieve the desired goal.

Modelling the network of agents aims at achieving scheduled orchestration between sensors and actuators' events. Sensors are passive agents; they are generally used to monitor and report quantitative measurements observed in perception and information fusion phases. Controllers inquire simple knowledge representation and symbolic rules to perform their reasoning tasks for planning and communicating within their surroundings [5]. The agent-based modelling approaches interpret heterogeneity, autonomy, social dynamics, and interactions between agents or distributed computational devices [7]. Each agent in the proposed infrastructure is considered a closed-loop feedback system; e.g., it is goal-oriented and self-directed, interacting with other agents to achieve its designated goal while adapting to changes in the surrounding environment.

3.2. The MARL trainer-executor component

The trainer-executor component is based on a multi-agent reinforcement learning approach. The **reinforcement learning (RL)** is one of the main Machine Learning (ML) paradigms that has recently witnessed rapid growth. Approaches of RL allow intelligent agent systems to learn a set of actions, which form a plan or a policy to interact with the environment. In ML, the learning process applies an inductive reasoning strategy, and in RL, specifically, this reasoning strategy tends to maximize a cumulative reward while learning the plan to accomplish a specific task [16]. In RL, knowledge is built over time, allowing the CPSs, i.e., robots, to adapt while interacting with the environment.

Typically, a single-agent problem is formulated as a Markov Decision problem defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma \rangle$, where an agent observes a state $s \in \mathcal{S}$ and selects an action $a \in \mathcal{A}$. The state transition $\mathcal{T}$ is defined by the probability function $P(s'|s,a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$ which receives a reward $\mathcal{R}$ defined by $r(s,a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$. The goal is to maximize the expected cumulative discounted return defined by $\mathcal{R}_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, where $\gamma \in [0,1]$ is a discount factor that balances between immediate and future rewards.

Methods of **Deep RL (DRL)** further inherit the deep learning capability that efficiently allows the discovery of features belonging to the learned policy. In DRL methods, neural networks learn to approximate either an optimal policy or a value function. The former, deep RL, follows a policy gradient approach that learns a parametrized policy $\pi(s,u)$ $V^{\pi}(s) = \mathbb{E}_{\pi}[\mathcal{R}_t | s_t = s]$ [30]. The latter, neural networks, learns a value function $Q(s,a)$ for (s: state, a: action) pairs, and the policy is formed by the selected actions that maximize the reward $Q^{\pi}(s,a) = E_{\pi}[\mathcal{R}_t | s_t = s, a_t = a]$ [31]. Both learning methods are combined to form a more advanced so-called actor-critic technique [28], where the actor is the learned policy, and the critic refers to the systems. The critic part of the network provides the rewards to the agents such that a single agent learns how to interact with its surrounding dynamically changing environment. This actor-critique technique forms a DRL framework that aims at learning the optimal policy an agent would follow to interact with its environment.

Multi-agent RL (MARL) approaches extend the single-agent RL by including distributed decision-making at a larger scale. Multiple agents are generally trained to execute strategies in a decentralized manner [26]. In MARL, the problem formulation is modelled as a partially observable Markov decision problem within a DRL framework [20]. However, the learning task becomes difficult when the environment is dynamic. In this case, each agent in a multi-agent system often assumes all other learning agents as part of the environment leading to a non-stationary situation. This situation may lead to choosing an optimal policy that might not be robust since it is exerted distributively. To mitigate the problem of non-stationarity, several approaches for centralized training with decentralized execution have been proposed [19, 11]. Nevertheless, centralized parts can destroy the benefits of having decentralized environments. Including a centralized processing unit while learning, often called a critic network, trained to minimize the loss in actor-critic methods, solves the non-stationarity issue by ensuring joint observability of the global state. This unit is then removed during the execution and once individual agents learn the optimized policy via policy gradient in a decentralized manner. However, a centralized solution is not suited for scalable problems, especially when the action

space grows exponentially with the number of agents involved in the decision-making and the environment changes constantly. Several approaches mitigate the scalability issue. These approaches vary between shared critic networks for actor-critic algorithms, such as MADDPG [19] and QMIX [22]. In MADDPG, the learning policies are based on multi-agent coordination that trains on an ensemble of policies leading to more robust multi-agent policies. On the other hand, the QMIX employs a network that estimates joint action values. These joint actions are considered monotonic per agent to guarantee tractability and consistency between centralized and decentralized policies. Some other methods use communication across the network of multi-agents, such as DIAL [11] and CommNet [27] to solve the problem of scalability. The DIAL implements deep neural networks to learn end-to-end communication protocols in complex environments with partial observability. In this approach, the agents backpropagate error derivatives using a centralized learning approach with decentralized execution. The COMNet is a neural network model where the agents learn how to communicate continuously to perform cooperative tasks. Other methods, such as the macro-actions approach introduced in [2], provide abstractions to improve scalability. These abstractions seek to remove unnecessary details in a trained multi-agent model, thereby demonstrating scalability in several domains. However, this scalability introduces other open questions and challenges to solve, such as sparse and delayed rewards where many actions are to be taken before a reward signal is available. Other problems are due to the combinatorial nature of MARL as a result of the exponential growth of the agents' action spaces.

MARL systems are adopted in competitive, cooperative, and mixed scenarios. They have been proven very useful for building effective decision-making for systems within several areas, such as for example, analysing, learning, and modelling:

- Analysis of behaviours through the evaluation of agent's algorithms in multi-agent scenarios. These methods are generally used in game theory (e.g., Atari, social dilemmas).
- Learning to communicate with other agents through protocols. With effective decision-making, these protocols solve cooperative tasks.
- Learning to cooperate using actions and local observations together with effective decision-making.
- Modelling other agents through learning to reason how other agents accomplish their tasks. The reasoning is carried out by effective decision-making.

MARL using deep learning requires a tremendous number of samples for effective training and robust performance.

As with any other ML approach, the DRL must have a trainer-executor component [21] to train a model deciding on the set of actions that must be taken and executing them while interacting within an environment. The essential constituents of this DRL are 1) a **trainer** that is a collection of agent learners responsible for sampling the data from the dataset and updating individual agent parameters and 2) an **executor** that forms a collection of agents/actors interacting with the environment. The results of each agent's training including all sets of actions along with their corresponding cost/reward are sent to the MARL using thread programming. Those results are tuples in the format of $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma \rangle$ per agent, where a state $s \in \mathcal{S}$ can be for example the agent's geo-position (e.g. (3,4), (4,5), (5,6)), an action $a \in \mathcal{A}$ is the action the agent can take such as, go forward (GF), turn left (TL), turn right (TR), and go backward (GB). Each of the listed actions affects the geo-position state of the agent and is associated with a cost/reward integer value (e.g. -1, 3, 5), where the higher the value the closer the agent is to achieving its goal. Accordingly, actions with higher reward values are chosen by the agents. The MARL trainer-executor is based on DRL and can implement three types of architectures: centralized, decentralized, and networked architecture.

A centralized architecture depends on a central computing capability with complete knowledge about individual agents in the system and their policies. This central unit aims at coordinating and integrating the actions performed by the agents. It realizes the plan and actions required to complete the task safely and optimally, yet it is hard to scale. In a decentralized agent architecture, individual agents are responsible for their own observations, decisions, and actions. In a decentralized agent architecture, each agent works in a closed loop and is assigned a small task or goal to be reached. Agents in this setup are likely to follow a swarm algorithm to reach a global goal. A networked architecture, however, mitigates the realized problems from both architectures. This networked architecture defines a communication topology that enables connected agents to exchange the necessary information to interact with their environment [9]. Fig. 1 illustrates that the adopted architecture in this proposed framework is a mix of a centralized architecture and a networked MARL architecture. Interconnected agents form a group that shares necessary infor-

mation through communication channels. Adopting a hybrid architecture guarantees the system's scalability since faster communication between connected agents is ensured by minimizing the communication load from or to the MARL trainer-executor component. The MARL trainer-executor component is the central unit, see Fig. 1, that defines approaches for the training and execution of the MARL. It is responsible for controlling and orchestrating the information transfer between agents. In particular, it ensures a proper transfer of the learned policy between agents. The policy is transferred between the agents, especially from those agents leaving the network to those agents newly attached to the volatile network infrastructure. The MARL agents in this infrastructure are trained to provide coordinated tasks, and the task-sharing capability is enabled by a communication protocol agreed upon between the different agents.

3.3. The environment interface component

The environment interface component shapes the system's interaction with its surrounding environment. This interface has four constituents: 1) dataset interface responsible for receiving input data and send data to the other agents, 2) observer-interpreter that observes the surrounding environment and interprets the input data captured by the sensors, 3) policy maker-action executor that sets the policy based on the training process and 4) a feedback loop that feeds information about output back to the agents. The four interfaces are generally attached to individual agents within the system to enable their collaboration and interaction.

*The **dataset*** interface is responsible for sharing all information necessary for the system to train, operate and interact with the environment. This information is a mix of current and historical data. The dataset encompasses: 1) data captured by the system, 2) information converted into analysis, 3) the training dataset, 4) learned information from the model during the training process, 5) decided upon policies and actions taken by the system, and 6) feedback provided in the training process and during the execution and interaction with the environment. This component is also responsible for orchestrating the heterogeneous data received and requested by the different system components. To access the data, a shared memory is a central component that allows agents in the infrastructure to reason, make decisions, and create their policies and actions. Furthermore, it includes the dataset used during the MARL training process. This dataset is gradually incremented through the feedback from the system's interaction with the surrounding environment.

*The **observer-interpreter*** interface is responsible for capturing raw data through the existing sensors of the agents, and processing the data for further interpretation of the situational awareness. At this stage, protocols for connections and data transfer to the network of agents are realized. The captured data is sent to the interpreter, who converts these data into knowledge that brings self-awareness to agents. The technologies involved are context-aware that allow the different components of the system to automatically obtain information from either other existing agents and the surrounding environment.

*The **policy maker-action executor*** interface derives and concludes the set of proposed actions that are taken by the system by incorporating DRL algorithms with the reasoning strategies and evaluating the alternatives. The **policies** are learned during the training process, and they create operational plans and missions to be executed by the individual agents in the system. The **action executor** interface is attached to actuators. The executor's main purpose is to achieve the decided plans and actions.

*The **feedback (reward/cost)*** loop is incorporated to observe the system behaviour closely. The computational resources and the physical system affect each other via this loop. The feedback plays an essential role in the two stages of the system: the training and the execution. It provides the reward to build up the learned knowledge and policies during the training process. At the same time, during the execution process, it ensures that the exerted actions are aligned with system goals.

4. Robustness Analysis with Multi-Agent Reinforcement Learning

The term robustness analysis (RA) is commonly coupled with system flexibility. RA aims at evaluating the correctness of the system's reasoning and reliability when interacting with unforeseen or erroneous inputs. The term RA

refers to methods for evaluating initial decision commitments under conditions of uncertainty, providing an operational measure of flexibility. The subsequent decisions are then implemented over time. Thus, RA starts with an initial model design and evolves it over time, i.e., when new decisions are made. These new decisions are included in the model design.

RA can be found in several contexts, systems and domains with different interpretations [14]. Among these, one usage is to quantifying the perturbations' impacts on the system's functionality [18]. Another usage of RA can be found through generating and dealing with a diversified set of hypotheses and models with a high degree of heterogeneity to confirm the output results of the system decisions or actions [24]. RA is also considered a tool that improves the system robustness by incorporating random variations in the problem formulation [3]. Random variations is dealing with unplanned and uncontrollable variations from a planned level of performance. The variations may arise from external influences (e.g. unpredicted situations) or they may be inherited in the policy learned (e.g. slight variations in the actions taken). This way of dealing with unpredicted situations can improve the reliability of the system design. RA is defined as a reasoning method for structuring problem situations and evaluating decisions that can be staged, sequentially. This allows correct functioning in the presence of data noise and environmental fluctuations. In this case, RA is used to measure the distinctions between the initial decision, i.e., commitment and acceptable options stored in the database, and evolving and changed decisions. It will be a chain of decisions, which can be interpreted to evaluate the decision made over time. Moreover, RA also evaluates the implications of those decisions on the system's stability without affecting its flexibility of interacting with the environment. With RA, ML algorithms can increase the probability of a particular action option by evaluating decisions that can be sequentially staged. When RA is adopted in ML, it can improve operations functionalities despite data noise and environmental fluctuations [14]. At the system design stage, RA imposes certain assumptions on the dynamicity of the environment. These assumptions consider the probabilistic nature of failures or unforeseen situations in the problem formulation [23]. At the execution stage, RA measures the degree to which the system can function with unforeseen or erroneous inputs when executing the set of actions decided by the reasoner and accordingly identifies vulnerability issues in the system behaviour. Based on this identification and to maintain the system's reliability, RA suggests robust solutions to execute rather than deterministic optimal ones [14].

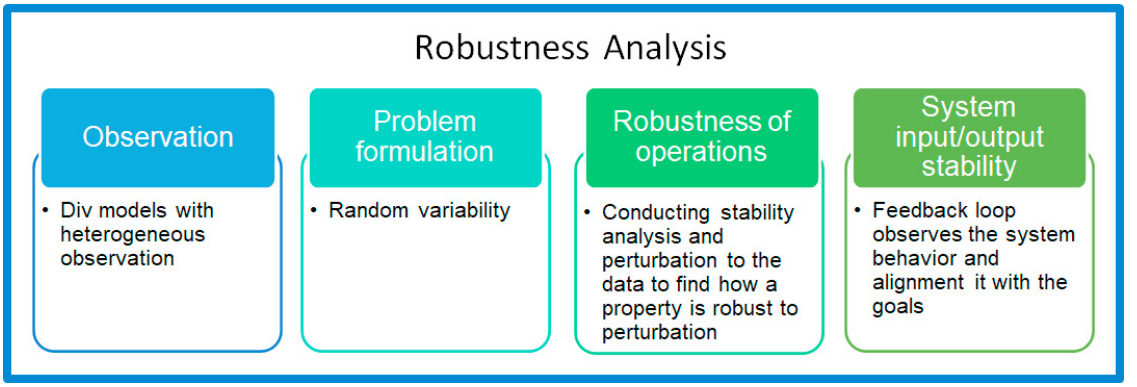

Fig. 2. Robustness analysis for the system's input-output stability

To guarantee the system flexibility and robust behaviour in a dynamically changing environment, a 4-step procedure that achieves RA, fig. 2, can be summarized as follows:

1. Generating diversified models with heterogeneous **observations** and assumptions to make a consensus about the observations or results [32, 24].
2. Incorporating random variability in the **problem formulation** [3].
3. Adding perturbation to the data to evaluate the **robustness of the operation** [18]
4. Seeking **input-output stability** of the overall system behaviour [23].

Robustness analysis is combined with multi-agent reinforcement learning to provide optimal solutions that are also robust solutions. The reinforcement learning is applied to observations made in the environment to get the most appropriate routes in a certain setting. On these routes, the robustness analysis is applied to find the most robust route.

The observations are concerned with the input data captured by the system's sensors. These captured data highly affect the decision-making of the system since they might be redundant, incorrect, and include noise. Typically, this data is processed by ML algorithms to identify information about the system status. Accordingly, robustness analysis should ensure the correct interpretation of the data, allowing, for instance, multiple sensors to observe the same phenomena to ensure high accuracy. Robustness analysis is applied by incorporating random variability into the problem formulation, see Fig. 2 (Robustness of operations). The reasoner evaluates a set of alternative actions before making the final decision into action and sending it to the CPSs. This evaluation involves verifying the prospective rewards of alternative decisions while considering the environment's dynamicity. The robustness of operations focuses on the execution of the set of actions decided by the system when interacting with the environment. To get the set of actions, the training use deep reinforcement learning, DRL. During the DRL training process, perturbations are added to the data in the dataset. This is used by robustness analysis to verify and validate the system's reliability. These perturbations ensure the system's stability by learning to react to unforeseen situations. In this case, RA considers additional assumptions on situations that likely can occur, providing stochastic features instead of deterministic ones. The system reasons about these stochastic features. Accordingly, RL decides on the set of possible actions and RA chooses the action that is more likely to maintain the system robustness according to a certain situation.

The overall system input-output stability is considered after the execution of the actions. At this stage, a feedback loop is incorporated into the CPS to observe the system's behaviour, closely. This feedback loop ensures that the exerted actions are aligned with the system goal. In addition, it continuously updates the rewards of the actions taken, incrementally building up the dataset for better future system behaviour. A successful increment of RA can improve the reliability of a system especially within a dynamic environment. RA should be applied to the entire CPS infrastructure to ensure robustness.

5. Robustness Analysis in RAMARL

5.1. Robustness analysis in the network of agents component

RA in the network of agents component of RAMARL applies sensor fusion methods on the **input data**, utilizing various and heterogeneous sensors to ensure the reliability of the data. Furthermore, RA applies concepts and approaches of fault tolerance, privacy, and security to ensure the stability and availability of the data captured from the sensors. On the **problem formulation**, RA, on one hand, extrapolates vast amounts of data to achieve high accuracy in forecasting and prediction. On the other hand, RA facilitates the attachment and detachment of agents to and from the network in a volatile manner, i.e., connect to and communicate with the agents that are affected by an action and disconnect the agents that are not affected. This volatility ensures a flexible problem formulation. Individual agents are by nature goal-oriented. Each applies its learned policy to achieve a designated goal. This decentralized goal task distribution ensures the **robustness of operations**, which is the third step of the RA procedure. RA achieves **input-output stability** by considering each agent as a closed-loop system; in other words, individual agents adapt their actions based on the interactions with other agents and the surrounding environment.

5.2. Robustness analysis in the MARL trainer-executor component

RA on the **input data** ensures an incremental building of the training dataset, even during the execution while collecting feedback from the system's interaction with the environment. The RA of the **problem formulation** adds random variability to secure the reliability of the system operations. **Robustness of operations** is ensured when individual agents learn the policy in the training process. RA safeguards the proper transfer of the learned policies between agents. The overall system **input-output stability** is guaranteed by updating the learned policies overtime during the training as well as the execution processes.

5.3. Robustness analysis in the environment component

Robustness analysis to deal with the **input data** applies a diversified set of observation tools (e.g., sensors) to ensure the robustness of the data. When multiple sensors make observations, a conflict resolution system manages

data ambiguity. In addition, RA relies on context-aware technologies to ensure the correctness of the captured data and situational awareness. Here the saying of "the larger the training dataset, the better the system performs" does not apply, especially when it needs to adapting the learned policies while facing unforeseen situations. During the training process, the agents learn in a distributed manner to formulate their policy, hence adopting RA in the **problem formulation**. It is the trainer's responsibility through the use of a large dataset to add and cover different scenarios to ensure system stability when unforeseen situations occur. The **robustness of operations** ensures the orchestration of storing and retrieving the necessary information for the system to operate. Policymaking is exerted during the training process. Individual agents update their policies depending on the dynamicity of the environment to align with the system's end goal.

The overall system **input-output stability** is guaranteed by applying continuous feedback during the training and execution processes. This feedback enables the system to observe its performance while interacting with the surrounding environment. Updates on policies and system behaviour are frequently sent to the database to build up the dataset for training incrementally.

5.4. Evaluation and discussion

A RAMARL infrastructure is formed by a network of agents having distributed goals and performing tasks in a decentralized manner. Accordingly, each agent is considered a closed-loop while communicating observed data, learned information, and actions taken through the MARL trainer-executor using thread programming. The trainer-executor learns the policy that is formed by the selected agents' actions that maximize the cumulative discounted reward given by $Q^{\pi}(s,a) = E_{\pi}[\mathcal{R}_t | s_t = s, a_t = a]$, for ($s$: state, a: action) pairs that is provided through a vast amount of data to ensure reliability and robustness.

The data set of a given agent is provided in the format of geo-positions (representing the agent's states), actions (e.g., GF-go forward, TL-turn left, TR-turn right), and an integer value indicating the reward/cost for the entire path, given by using reinforcement learning: ((1,2), (3,4), (4,5), (5,6), GF, 3), ((1,2), (2,3), (3,4), (4,5), (5,6), GL, -1), ((1,2), (4,5), (5,6), (GF, TF), 4), ((1,2) (2,3), (3,3), (3,5), (5,6), TL, 0).

These paths with rewards from each agent are stored in a database in RAMARL, which becomes data set for robust analysis. The MARL trainer-executor takes the results from each agent involved in the environment and calculates a result using tuples:

MARL tuples $\langle \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$ for this example are given as follows: $\mathcal{S}$ (set of states): [[(1,2), (3,4), (4,5), (5,6))], [(1,2), (2,3)], (3,4), (4,5), (5,6)], [(1,2), (4,5), (5,6)], [(1,2), (2,3), (3,3), (3,5], (5,6)] $\mathcal{A}$ (actions): Go-forward, Turn-Left, Turn-Right

The result of maximizing the cumulative discounted reward is inferred from the reward list given by $\mathcal{R}$ (reward): 3, -1, 4, 0. Accordingly, the set of selected states for this agent is = [(1,2), (4,5), (5,6)] suggesting (GF, TF or TR) as the set of actions that need to be taken by the agent.

The RAMARL uses MARL trainer-executor to calculate a robust outcome from all the agents taking the same route in the environment, which are connected to the RAMARL. RAMARL includes robustness analysis to get a robust decision. This decision is considered to be the most robust outcome of the agents, at hand, since it is based on several different agents' performances. Thus, the more agents that come up with the same path with good or acceptable rewards the better the outcome is for RAMARL.

6. Conclusions and further work

This paper presents RAMARL, a system that combines robustness analysis with multi-agent reinforcement learning to reason robustly in a dynamically changing environment. The combination strengthens the agents' reinforcement learning, increasing the reliability and flexibility of the system by applying robustness analysis. When robustness analysis is incorporated into the system, the analysis suggests robust solutions and actions. Those suggested actions ensure the system's overall reliability and flexibility. Accordingly, by taking those proposed actions, the system adapts to the dynamicity of the surroundings hence allowing the system to have a robust behaviour that is optimal according to a specific situation. This additional adaptability is opposed to making deterministic optimal decisions as provided by reinforcement learning when adopted alone by the system and which may not lead to robust behaviour.

The next step is to test the RAMARL system and study the results in a real environment. The expectation is the see that the agents' behaviour is robust which may or may not be optimal.

References

[1] Ahmadzai Ahmadi, Chantal Cherifi, Vincent Cheutet, and Yacine Ouzrout. A review of cps 5 components architecture for manufacturing based on standards. In *2017 11th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*, pages 1–6. IEEE, 2017.

[2] Christopher Amato, George Konidaris, Leslie P Kaelbling, and Jonathan P How. Modeling and planning with macro-actions in decentralized pomdps. *Journal of Artificial Intelligence Research*, 64:817–859, 2019.

[3] Christian Bucher. Robustness analysis in structural optimization. *Structure and Infrastructure Engineering*, 5(4):287–293, 2009.

[4] Alejandro Chacón, Cecilio Angulo, and Pere Ponsa. Developing cognitive advisor agents for operators in industry 4.0. *New Trends in the Use of Artificial Intelligence for the Industry 4.0*, page 127, 2020.

[5] Barbara Dunin-Keplicz and Andrzej Szałas. Agents in approximate environments. In *Games, Actions and Social Software*, pages 141–163. Springer, 2012.

[6] Barbara Dunin-Keplicz and Rineke Verbrugge. *Teamwork in multi-agent systems: A formal approach*, volume 21. John Wiley & Sons, 2011.

[7] Joshua M Epstein. Agent-based computational models and generative social science. *Complexity*, 4(5):41–60, 1999.

[8] Ronald Fagin, Yoram Moses, Joseph Y Halpern, and Moshe Y Vardi. *Reasoning about knowledge*. MIT press, 2003.

[9] Tingxiang Fan, Pinxin Long, Wenxi Liu, and Jia Pan. Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios. *The International Journal of Robotics Research*, page 0278364920916531, 2020.

[10] Michael J Fischer and Richard E Ladner. Propositional dynamic logic of regular programs. *Journal of computer and system sciences*, 18(2):194–211, 1979.

[11] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

[12] Michał Grabowski and Andrzej Szałas. A technique for learning similarities on complex structures with applications to extracting ontologies. In *International Atlantic Web Intelligence Conference*, pages 183–189. Springer, 2005.

[13] Anne Håkansson. Ipsum–an approach to smart volatile ict-infrastructures for smart cities and communities. *Procedia computer science*, 126:2107–2116, 2018.

[14] Anne Håkansson, Aya Saad, Akhil Anand, Vilde Gjærum, Haakon Robinson, and Katrine Seel. Robust reasoning for autonomous cyber-physical systems in dynamic environments. *Procedia Computer Science*, 192:3966–3978, 2021.

[15] David Harel, Dexter Kozen, and Jerzy Tiuryn. Dynamic logic. In *Handbook of philosophical logic*, pages 99–217. Springer, 2001.

[16] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797, 2019.

[17] Fei Hu, Yu Lu, Athanasios V Vasilakos, Qi Hao, Rui Ma, Yogendra Patil, Ting Zhang, Jiang Lu, Xin Li, and Neal N Xiong. Robust cyber–physical systems: Concept, models, and implementation. *Future generation computer systems*, 56:449–475, 2016.

[18] Wiktor Jakowluk and Karol Godlewski. Robustness analysis of the estimators for the nonlinear system identification. *Entropy*, 22(8):834, 2020.

[19] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[20] Praveen Palanisamy. Multi-agent connected autonomous driving using deep reinforcement learning. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2020.

[21] Arnu Pretorius, Kale ab Tessera, Andries P. Smit, Kevin Eloff, Claude Formanek, St John Grimbly, Siphelele Danisa, Lawrence Francis, Jonathan Shock, Herman Kamper, Willie Brink, Herman Engelbrecht, Alexandre Laterre, and Karim Beguir. Mava: A research framework for distributed multi-agent reinforcement learning. *arXiv preprint arXiv:2107.01460*, 2021.

[22] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304. PMLR, 2018.

[23] Matthias Rungger and Paulo Tabuada. A notion of robustness for cyber-physical systems. *IEEE Transactions on Automatic Control*, 61(8):2108–2123, 2015.

[24] Jonah N Schupbach. Robustness analysis as explanatory reasoning. *The British Journal for the Philosophy of Science*, 69(1):275–300, 2018.

[25] Dimitrios Serpanos. The cyber-physical systems revolution. *Computer*, 51(3):70–73, 2018.

[26] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

[27] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. *Advances in neural information processing systems*, 29, 2016.

[28] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[29] Ngoc-Hien Tran, Hong-Seok Park, Quang-Vinh Nguyen, and Tien-Dung Hoang. Development of a smart cyber-physical manufacturing system in the industry 4.0 context. *Applied Sciences*, 9(16):3325, 2019.

[30] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[31] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[32] Michael Weisberg. Robustness analysis. *Philosophy of science*, 73(5):730–742, 2006.

[33] Li Da Xu and Lian Duan. Big data for cyber physical systems in industry 4.0: a survey. *Enterprise Information Systems*, 13(2):148–169, 2019.