

A FRAMEWORK TO MEASURE THE ROBUSTNESS OF PROGRAMS IN THE UNPREDICTABLE ENVIRONMENT*

VALENTINA CASTIGLIONI ^a, MICHELE LORETI ^b, AND SIMONE TINI ^c

^a Reykjavik University, Iceland
e-mail address: valentinac@ru.is, vale.castiglioni@gmail.com

^b University of Camerino, Italy
e-mail address: michele.loreti@unicam.it

^c University of Insubria, Italy
e-mail address: simone.tini@uninsubria.it

ABSTRACT. Due to the diffusion of IoT, modern software systems are often thought to control and coordinate smart devices in order to manage assets and resources, and to guarantee efficient behaviours. For this class of systems, which interact extensively with humans and with their environment, it is thus crucial to guarantee their “correct” behaviour in order to avoid unexpected and possibly dangerous situations. In this paper we will present a framework that allows us to measure the robustness of systems. This is the ability of a program to tolerate changes in the environmental conditions and preserving the original behaviour. In the proposed framework, the interaction of a program with its environment is represented as a sequence of random variables describing how both evolve in time. For this reason, the considered measures will be defined among probability distributions of observed data. The proposed framework will be then used to define the notions of adaptability and reliability. The former indicates the ability of a program to absorb perturbation on environmental conditions after a given amount of time. The latter expresses the ability of a program to maintain its intended behaviour (up-to some reasonable tolerance) despite the presence of perturbations in the environment. Moreover, an algorithm, based on statistical inference, is proposed to evaluate the proposed metric and the aforementioned properties. We use two case studies to describe and evaluate the proposed approach.

1. INTRODUCTION

With the advent of *IoT era* [Kop11] we witnessed to the diffusion of a plethora of *smart interconnected devices* that permit *sensing data* from the *environment* and to *operate* on it. This revolution stimulated the design of increasingly pervasive systems, often named *Cyber-Physical Systems* (CPSs) [RLSS10], that are thought to control and coordinate these devices in order to manage assets and resources, and to guarantee efficient behaviours. In particular, it is crucial to guarantee that these devices always behave “correctly” in their

Key words and phrases: Robustness, Adaptability, Reliability, Uncertain environment.

* A preliminary version of this paper appeared as [CLT21].

interaction with humans and/or other devices, in order to avoid unexpected and potentially dangerous situations.

Considering that the behaviour of a program controlling the device is largely determined by its extensive interplay with the surrounding environment, this means that the design of CPSs must include some formal tool allowing us to describe how the environment evolves and, if needed, how it reacts to the actions of the program. For instance, while programming an unmanned vehicle (UV) that has to autonomously set its trajectory to avoid obstacles, we have to consider the possible wind (in case of an aerial UV), or stream (for an underwater UV) scenarios describing the conditions under which the UV will operate. However, given the dynamical and, sometimes, unpredictable behaviour of the phenomena that constitute the environment, these scenarios cannot be fully deterministic. Instead, the *evolution of the environment* should be described in terms of a *stochastic process* that gives, at each step, a probabilistic measure of the *observed dynamics* of the environment, including the likelihood of events that could constitute a hazard to our device, like, e.g., a gust of wind in the case of an aerial UV. Being the dynamics continuous, it is natural for the stochastic process to be defined over a *continuous state space*. At the same time, the program controlling the system might incorporate *discrete random behaviours* to mitigate the uncertainty in the environmental conditions. As a consequence, the interaction between the program and the environment will be represented in terms of a stochastic process whose states describe how the environmental conditions and the state of the program evolve along a computation.

In the literature, we can find a wealth of proposals of stochastic and probabilistic models, as, e.g., Probabilistic Automata [Seg95], Stochastic Process Algebras [HHH⁺94, CH08, BNL13], Labelled Markov Chains and Stochastic Hybrid Systems [CLe07, HLS00], *Markov Decision Processes* [Put05], and *ad hoc* solutions for specific application contexts, as, e.g., establishing safety guarantees for drones flying under particular circumstances [VNGV16, HJS⁺14]. Yet, in these studies, either the environment is not explicitly taken into account, or it is modelled only deterministically. In addition to that, due to the variety of applications and heterogeneity of systems, no general formal framework to deal with these challenges has been proposed so far. The lack of concise abstractions and of an automatic support makes the analysis and verification of the considered systems difficult, laborious, and error prone. Hence, as a first contribution, in this paper we propose a semantic framework allowing us to model the behaviour of CPSs taking into account the continuous dynamics of the environment.

We can then focus on the analysis of the obtained behaviour. One of the main challenges here is to quantify how *variations in the environmental conditions* can affect the *expected behaviour* of the system. In this paper we will present a framework that allows us to *measure the robustness* of systems. This is the ability of a program to *tolerate* changes in the environmental conditions and preserving the original behaviour. Since the interaction of the program with the enclosing environment is represented as a sequence of random variables, the considered measures will be defined among probability distributions of observed data. The proposed framework will then be used to define the robustness notions of *adaptability* and *reliability*. As an example, if we consider the above mentioned aerial UV scenario, a perturbation can be caused by a gust of wind that moves the UV out of its trajectory. We will say that the program controlling the UV is *adaptable* if it can retrieve the initial trajectory within a suitable amount of time. In other words, we say that a program is adaptable if no matter how much its behaviour is initially affected by the perturbations, it is able to react to them and regain its intended behaviour within a given amount of time. On

the other hand, it may be the case that the UV is able to detect the presence of a gust of wind and can oppose to it, being only slightly moved from its initial trajectory. In this case, we say that the program is *reliable*. Hence, *reliability* expresses the ability of a program to maintain its intended behaviour (up-to some reasonable tolerance) despite the presence of perturbations in the environment.

Detailed outline of paper contributions. In this paper we first propose a semantic framework that permits modelling the behaviour of programs operating in an evolving environment. In the proposed approach, the interaction among these two elements is represented in a purely *data-driven* fashion. In fact, while the environmental conditions are (partially) available to the program as a set of data, allowing it to adjust its behaviour to the current situation, the program is also able to use data to (partially) control the environment and fulfil its tasks. The program-environment interplay is modelled in terms of the changes they induce on a set of application-relevant data, henceforth called *data space*. This approach will allow for a significant simplification in modelling the behaviour of the program, which can be *isolated* from that of the environment. Moreover, as common to favour *computational tractability* [ADB11, AKLP10], a *discrete time* approach is adopted.

In our model, a *system* consists in *three distinct components*:

- (1) a *process* P describing the behaviour of the program,
- (2) a *data state* \mathbf{d} describing the current state of the data space, and
- (3) an *environment evolution* \mathcal{E} describing the effect of the environment on \mathbf{d} .

As we focus on the interaction with the environment, we abstract from the internal computation of the program and model only its activity on \mathbf{d} . At each step, a process can *read/update* values in \mathbf{d} and \mathcal{E} applies on the resulting data state, providing a new data state at the next step. To deal with the uncertainties, we introduce *probability* at two levels:

- (i) we use the *discrete generative probabilistic model* [vGSS95] to define processes, and
- (ii) \mathcal{E} induces a *continuous distribution* over data states.

The behaviour of the system is then entirely expressed by its *evolution sequence*, i.e., the sequence of distributions over data states obtained at each step. Given the novelties of our model, as a side contribution we show that this behaviour defines a *Markov process*.

Remark 1.1. We remark that it is not possible to replace the continuous distribution induced by \mathcal{E} with a discrete one without introducing limiting assumptions on the behaviour of the environment. In fact, in the continuous setting, each data state in the support of the distribution is potentially reachable. Conversely, when a discrete (or finite state) setting is considered, it is necessary to partition the data space, so that some data states have to be identified in a single state, or they simply become unreachable. Hence, a simplification to the discrete setting would result in a loss of information on the behaviour of the environment, thus making the model unfeasible to deal with the slight modifications on data induced by the uncertainties.

The second contribution is the definition of a *metric semantics* in terms of a (time-dependent) distance on the evolution sequences of systems, which we call the *evolution metric*. The *evolution metric* will allow us to:

- (1) verify how well a program is fulfilling its tasks by comparing it with its specification,
- (2) compare the activity of different programs in the same environment,

- (3) compare the behaviour of one program with respect to different environments and changes in the initial conditions.

The evolution metric will consist of two components: a *metric on data states* and the *Wasserstein metric* [Vas69]. The former is defined in terms of a (time-dependent) *penalty function* allowing us to compare two data states only on the base of the objectives of the program. The latter lifts the metric on data states to a metric on distributions on data states. We then obtain a metric on evolution sequences by considering the maximal of the Wasserstein distances over time. The proposed metric is then used to formalise the notions of *robustness*, *adaptability*, and *reliability*.

Remark 1.2. Notice that the analysis of behaviour presented in items (2) and (3) above are made possible by our choice of deriving the Markov process from the specification of the program P and the environment \mathcal{E} , instead of taking it as given. Hence, it is enough for us to substitute the specification of a process (respectively, the environment) with another one, and consider the evolution sequence induced by it to obtain the new behaviour. This is what we do, for instance, in the case study in Section 7, where we compare a genuine program with a compromised one working in the same environment.

A *randomised algorithm* that permits estimating the evolution sequences of systems and thus for the evaluation of the evolution metric is then introduced. Following [TK10], the Wasserstein metric is evaluated in time $O(N \log N)$, where N is the (maximum) number of samples. We already adopted this approach in [CLT20a] in the context of finite-states self-organising collective systems, without any notion of environment or data space. Moreover, we will also show how the proposed algorithm can be used to estimate the *robustness*, *adaptability*, and *reliability*, of a system.

Organisation of contents. After reviewing some background mathematical notions in Section 2, we devote Section 3 to the presentation of our *model*. In Section 4 we discuss the *evolution metric* and all the ingredients necessary to define it. Then, in Section 5 we provide the *algorithm*, based on statistical inference, for the evaluation of the metric. The notions of *adaptability* and *reliability* of programs are formally introduced in Section 6 together with an example of application of our algorithm to their evaluation. In these sections a running scenario is used to clarify the proposed approach. This consists of a stochastic variant of the three-tanks laboratory experiment described in [RKO⁺97]. In Section 7 our methodology is used to analyse the *engine system* proposed in [LMMT21]. This consists of two supervised self-coordinating refrigerated engine systems that are subject to cyber-physical attacks aiming to inflict *overstress of equipment* [GGI⁺15]. Finally, Section 8 concludes the paper with a discussion on related work and future research directions.

2. BACKGROUND

In this section we introduce the mathematical background on which we build our contribution. We remark that we present in detail only the notions that are fundamental to allow any reader to understand our work, like, e.g., those of probability measure and metric. Other notions that are needed exclusively to guarantee that the Mathematical constructs we use are well-defined (e.g., the notion of Radon measure for Wasserstein hemimetrics) are only mentioned. The interested reader can find all the formal definitions in any Analysis textbook.

Measurable spaces and measurable functions. A σ -algebra over a set Ω is a family Σ of subsets of Ω such that: (1) $\Omega \in \Sigma$, (2) Σ is closed under complementation, and (3) Σ is closed under countable union. The pair (Ω, Σ) is called a *measurable space* and the sets in Σ are called *measurable sets*, ranged over by $\mathbb{A}, \mathbb{B}, \dots$

For an arbitrary family Φ of subsets of Ω , the σ -algebra *generated* by Φ is the smallest σ -algebra over Ω containing Φ . In particular, we recall that given a topology T over Ω , the *Borel σ -algebra* over Ω , denoted $\mathcal{B}(\Omega)$, is the σ -algebra generated by the open sets in T . For instance, given $n \in \mathbb{N}^+$, $\mathcal{B}(\mathbb{R}^n)$ is the σ -algebra generated by the open intervals in \mathbb{R}^n .

Given two measurable spaces (Ω_i, Σ_i) , $i = 1, 2$, the *product σ -algebra* $\Sigma_1 \otimes \Sigma_2$ is the σ -algebra on $\Omega_1 \times \Omega_2$ generated by the sets $\{\mathbb{A}_1 \times \mathbb{A}_2 \mid \mathbb{A}_i \in \Sigma_i\}$.

Given measurable spaces $(\Omega_1, \Sigma_1), (\Omega_2, \Sigma_2)$, a function $f: \Omega_1 \rightarrow \Omega_2$ is said to be Σ_1 -*measurable* if $f^{-1}(\mathbb{A}_2) \in \Sigma_1$ for all $\mathbb{A}_2 \in \Sigma_2$, with $f^{-1}(\mathbb{A}_2) = \{\omega \in \Omega_1 \mid f(\omega) \in \mathbb{A}_2\}$.

Probability spaces and random variables. A *probability measure* on a measurable space (Ω, Σ) is a function $\mu: \Sigma \rightarrow [0, 1]$ such that $\mu(\Omega) = 1$, $\mu(\mathbb{A}) \geq 0$ for all $\mathbb{A} \in \Sigma$, and $\mu(\bigcup_{i \in I} \mathbb{A}_i) = \sum_{i \in I} \mu(\mathbb{A}_i)$ for every countable family of pairwise disjoint measurable sets $\{\mathbb{A}_i\}_{i \in I} \subseteq \Sigma$. Then (Ω, Σ, μ) is called a *probability space*. We let $\Pi(\Omega, \Sigma)$ denote the set of all probability measures over (Ω, Σ) .

For $\omega \in \Omega$, the *Dirac measure* δ_ω is defined by $\delta_\omega(\mathbb{A}) = 1$, if $\omega \in \mathbb{A}$, and $\delta_\omega(\mathbb{A}) = 0$, otherwise, for all $\mathbb{A} \in \Sigma$. Given a countable set of reals $(p_i)_{i \in I}$ with $p_i \geq 0$ and $\sum_{i \in I} p_i = 1$, the *convex combination* of the probability measures $\{\mu_i\}_{i \in I} \subseteq \Pi(\Omega, \Sigma)$ is the probability measure $\sum_{i \in I} p_i \cdot \mu_i$ in $\Pi(\Omega, \Sigma)$ defined by $(\sum_{i \in I} p_i \cdot \mu_i)(\mathbb{A}) = \sum_{i \in I} p_i \mu_i(\mathbb{A})$, for all $\mathbb{A} \in \Sigma$. A probability measure $\mu \in \Pi(\Omega, \Sigma)$ is called *discrete* if $\mu = \sum_{i \in I} p_i \cdot \delta_{\omega_i}$, with $\omega_i \in \Omega$, for some countable set of indexes I . The *support* of a discrete probability measure μ is defined as $\text{supp}(\mu) = \{\omega \in \Omega \mid \mu(\omega) > 0\}$.

Assume a probability space (Ω, Σ, μ) and a measurable space (Ω', Σ') . A function $X: \Omega \rightarrow \Omega'$ is called a *random variable* if it is Σ -measurable. The *distribution measure*, or *cumulative distribution function* (cdf), of X is the probability measure μ_X on (Ω', Σ') defined by $\mu_X(\mathbb{A}) = \mu(X^{-1}(\mathbb{A}))$ for all $\mathbb{A} \in \Sigma'$. Given random variables X_i from $(\Omega_i, \Sigma_i, \mu_i)$ to (Ω'_i, Σ'_i) , $i = 1, \dots, n$, the collection $\mathbf{X} = [X_1, \dots, X_n]$ is called a *random vector*. The cdf of a random vector \mathbf{X} is given by the *joint* distribution of the random variables in it.

Notation 2.1. With a slight abuse of notation, in the examples and explanations throughout the paper, we will sometimes use directly the cdf of a random variable rather than formally introducing the probability measure defined on the domain space. In fact, since we will consider Borel sets over \mathbb{R}^n , for some $n \geq 1$, it is usually easier to define the cdfs than the probability measures on them. Similarly, when the cdf is absolutely continuous with respect to the Lebesgue measure, then we shall reason directly on the *probability density function* (pdf) of the random variable, namely the Radon-Nikodym derivative of the cdf with respect to the Lebesgue measure.

Consequently, we shall also henceforth use the more suggestive term **distribution** in place of the terms probability measure, cdf and pdf.

The Wasserstein hemimetric. A *metric* on a set Ω is a function $m: \Omega \times \Omega \rightarrow \mathbb{R}^{\geq 0}$ s.t. $m(\omega_1, \omega_2) = 0$ iff $\omega_1 = \omega_2$, $m(\omega_1, \omega_2) = m(\omega_2, \omega_1)$, and $m(\omega_1, \omega_2) \leq m(\omega_1, \omega_3) + m(\omega_3, \omega_2)$, for all $\omega_1, \omega_2, \omega_3 \in \Omega$. We obtain a *pseudometric* by relaxing the first property to $m(\omega_1, \omega_2) = 0$ if $\omega_1 = \omega_2$. Then, a *hemimetric* is a pseudometric that is not necessarily symmetric. A

(pseudo-, hemi-)metric m is l -bounded if $m(\omega_1, \omega_2) \leq l$ for all $\omega_1, \omega_2 \in \Omega$. For a (pseudo-, hemi-)metric on Ω , the pair (Ω, m) is a *(pseudo-, hemi-)metric space*.

Given a (pseudo-, hemi-)metric space (Ω, m) , the (pseudo-, hemi-)metric m induces a natural topology over Ω , namely the topology generated by the open ε -balls, for $\varepsilon > 0$, $B_m(\omega, \varepsilon) = \{\omega' \in \Omega \mid m(\omega, \omega') < \varepsilon\}$. We can then naturally obtain the Borel σ -algebra over Ω from this topology.

In this paper we are interested in defining a *hemimetric on distributions*. To this end we will make use of the Wasserstein lifting [Vas69], which evaluates the infimum, with respect to the joint distributions, of the expected value of the distance over the two distributions, and whose definition is based on the following notions and results. Given a set Ω and a topology T on Ω , the topological space (Ω, T) is said to be *completely metrisable* if there exists at least one metric m on Ω such that (Ω, m) is a complete metric space and m induces the topology T . A *Polish space* is a separable completely metrisable topological space. In particular, we recall that: (i) \mathbb{R} is a Polish space; and (ii) every closed subset of a Polish space is in turn a Polish space. Moreover, for any $n \in \mathbb{N}$, if $\Omega_1, \dots, \Omega_n$ are Polish spaces, then the Borel σ -algebra on their product coincides with the product σ -algebra generated by their Borel σ -algebras, namely

$$\mathcal{B}\left(\prod_{i=1}^n \Omega_i\right) = \prod_{i=1}^n \mathcal{B}(\Omega_i).$$

(This is proved, e.g., in [Bog07] as Lemma 6.4.2 whose hypothesis are satisfied by Polish spaces since they are second countable.) These properties of Polish spaces are interesting for us since they guarantee that all the distributions we consider in this paper are Radon measures and, thus, the Wasserstein lifting is well-defined on them. For this reason, we also directly present the Wasserstein hemimetric by considering only distributions on Borel sets.

Definition 2.2 (Wasserstein hemimetric). Consider a Polish space Ω and let m be a hemimetric on Ω . For any two distributions μ and ν on $(\Omega, \mathcal{B}(\Omega))$, the *Wasserstein lifting* of m to a distance between μ and ν is defined by

$$\mathbf{W}(m)(\mu, \nu) = \inf_{\mathfrak{w} \in \mathfrak{W}(\mu, \nu)} \int_{\Omega \times \Omega} m(\omega, \omega') d\mathfrak{w}(\omega, \omega')$$

where $\mathfrak{W}(\mu, \nu)$ is the set of the *couplings* of μ and ν , namely the set of joint distributions \mathfrak{w} over the product space $(\Omega \times \Omega, \mathcal{B}(\Omega \times \Omega))$ having μ and ν as left and right marginal, respectively, namely $\mathfrak{w}(\mathbb{A} \times \Omega) = \mu(\mathbb{A})$ and $\mathfrak{w}(\Omega \times \mathbb{A}) = \nu(\mathbb{A})$, for all $\mathbb{A} \in \mathcal{B}(\Omega)$.

Despite the original version of the Wasserstein distance being defined on a metric on Ω , the Wasserstein hemimetric given above is well-defined. We refer the interested reader to [FR18] and the references therein for a formal proof of this fact. In particular, the Wasserstein hemimetric is given in [FR18] as Definition 7 (considering the compound risk excess metric defined in Equation (31) of that paper), and Proposition 4 in [FR18] guarantees that it is indeed a well-defined hemimetric on $\Pi(\Omega, \mathcal{B}(\Omega))$. Moreover, Proposition 6 in [FR18] guarantees that the same result holds for the hemimetric $m(x, y) = \max\{y - x, 0\}$ which, as we will see, plays an important role in our work (cf. Definition 4.2 below).

Remark 2.3. As elsewhere in the literature, for simplicity and brevity, we shall henceforth use the term *metric* in place of the term hemimetric.

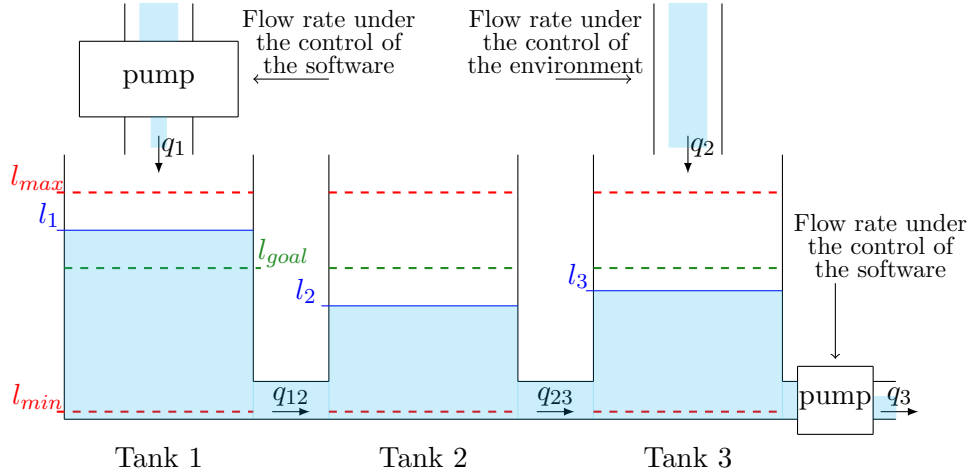


Figure 1. Schema of the three-tanks scenario.

3. THE MODEL

We devote this section to introduce the three components of our systems, namely the *data space* \mathcal{D} , the *process* P describing the behaviour of the program, and the *environment evolution* \mathcal{E} describing the effects of the environment. As already discussed in the Introduction, our choice of modelling program and environment separately will allow us to isolate programs from the environment and favour their analysis.

In order to help the reader grasp the details of our approach, we introduce a stochastic variant of the three-tanks laboratory experiment, that embodies the kind of program-environment interactions we are interested in. Several variants (with different number of tanks) of the n -tanks experiment have been widely used in control program education (see, among the others, [AL92, RKO⁺97, Joh00, ALGG⁺06]). Moreover, some recently proposed cyber-physical security testbeds, like *SWaT* [MT16, AGA⁺17], can be considered as an evolution of the tanks experiment. Here, we consider a variant of the three-tanks laboratory experiment described in [RKO⁺97].

Please notice that this example is meant to serve as a toy example allowing us to showcase the various features of our tool, without carrying out a heavy mathematical formulation. We delay to Section 7 the presentation of a case-study showing how our framework can be applied to the analysis of (more complex) real-world systems.

Notation 3.1. In the upcoming examples we will slightly abuse of notation and use a variable name x to denote all: the variable x , the (possible) function describing the evolution in time of the values assumed by x , and the (possible) random variable describing the distribution of the values that can be assumed by x at a given time. The role of the variable name x will always be clear from the context.

Example 3.2. As schematised in Figure 1, there are three identical tanks connected by two pipes. Water enters in the first and in the last tank by means of a pump and an incoming pipe, respectively. The last tank is equipped with an outlet pump. We assume that water flows through the incoming pipe with a rate q_2 that is determined by the environment, whereas the flow rates through the two pumps (respectively, q_1 and q_3) are under the control of the program controlling the experiment. The rates q_1, q_2, q_3 can assume values in the

range $[0, q_{max}]$, for a given a maximal flow rate q_{max} . The task of the program consists in guaranteeing that the levels of water in the three tanks fulfil some given requirements.

The level of water in tank i at time τ is denoted by $l_i(\tau)$, for $i = 1, 2, 3$, and is always in the range $[l_{min}, l_{max}]$, for suitable l_{min} and l_{max} giving, respectively, the minimum and maximum level of water in the tanks.

The principal difference between our version of the three-tanks and its classical formulation is the flow rate q_2 . In fact, we assume that q_2 cannot be controlled by the program, that can only indirectly observe the effects it has on l_3 and react to those. In our setting, this is equivalent to say that q_2 is under the control of the environment. Moreover, to obtain a more complex scenario, we assume that the value of q_2 (at a given time) can be described only probabilistically. This can be interpreted either as the effect of uncertainties (like, e.g., measurement errors of the sensors having to determine q_2), or as the consequence of an unpredictable behaviour of the environment. Consequently, the exact value of $q_2(\tau)$ is unknown and will be evident only at execution time. Still, we can consider different scenarios that render the assumptions we might have on the environment. For instance, we can assume that the flow rate of the incoming pipe is normally distributed with mean $q_{med} \in [0, q_{max}]$ and variance $\Delta_q > 0$, expressed, assuming sampling time interval $\Delta\tau$, by:

$$q_2(\tau + \Delta\tau) \sim N(q_{med}, \Delta_q) . \quad (3.1)$$

In a more elaborated scenario, we could assume that q_2 varies at each step by a value v that is normally distributed with mean 0 and variance 1. In this case, we have:

$$\begin{aligned} v(\tau) &\sim N(0, 1) \\ q_2(\tau + \Delta\tau) &= \min \{ \max \{ 0, q_2(\tau) + v(\tau) \}, q_{max} \} . \end{aligned} \quad (3.2)$$

Conversely, the two pumps are controlled by the program that, by reading the values of $l_1(\tau)$ and $l_3(\tau)$, can select the value of $q_1(\tau + \Delta\tau)$ and $q_3(\tau + \Delta\tau)$. In detail, the equations describing the behaviour of the program are the following:

$$q_1(\tau + \Delta\tau) = \begin{cases} \max\{0, q_1(\tau) - q_{step}\} & \text{if } l_1(\tau) > l_{goal} + \Delta_l \\ \min\{q_{max}, q_1(\tau) + q_{step}\} & \text{if } l_1(\tau) < l_{goal} - \Delta_l \\ q_1(\tau) & \text{otherwise;} \end{cases} \quad (3.3)$$

$$q_3(\tau + \Delta\tau) = \begin{cases} \min\{q_{max}, q_3(\tau) + q_{step}\} & \text{if } l_3(\tau) > l_{goal} + \Delta_l \\ \max\{0, q_3(\tau) - q_{step}\} & \text{if } l_3(\tau) < l_{goal} - \Delta_l \\ q_3(\tau) & \text{otherwise.} \end{cases} \quad (3.4)$$

Above, l_{goal} is the desired level of water in the tanks, while $q_{step} \in [0, q_{max})$ is the variation of the flow rate that is controllable by the pump in one time step $\Delta\tau$. The idea behind Equation (3.3) is that when l_1 is greater than $l_{goal} + \Delta_l$, the flow rate of the pump is decreased by q_{step} . Similarly, when l_1 is less than $l_{goal} - \Delta_l$, that rate is increased by q_{step} . The reasoning for Equation (3.4) is symmetrical. In both cases, we use the threshold $\Delta_l > 0$ to avoid continuous contrasting updates.

From now on, we will say that we are in *scenario 1* if q_2 is determined by Equation (3.1), and that we are in *scenario 2* in case q_2 is determined by Equation (3.2). In both scenarios, q_1 is determined by Equation (3.3) and q_3 by Equation (3.4).

The dynamics of $l_i(\tau)$ can be modelled via the following set of stochastic difference equations, with sampling time interval $\Delta\tau$:

$$\begin{aligned} l_1(\tau + \Delta\tau) &= l_1(\tau) + \Delta\tau \cdot (q_1(\tau) - q_{12}(\tau)) \\ l_2(\tau + \Delta\tau) &= l_2(\tau) + \Delta\tau \cdot (q_{12}(\tau) - q_{23}(\tau)) \\ l_3(\tau + \Delta\tau) &= l_3(\tau) + \Delta\tau \cdot (q_2(\tau) + q_{23}(\tau) - q_3(\tau)) \end{aligned} \quad (3.5)$$

where q_1, q_2, q_3 are as above (i.e., the flow rates through, respectively, the pump connected to the first tank, the incoming pipe, and the outlet pump), and q_{ij} denotes the flow rate from tank i to tank j . The evaluation of $q_{ij}(\tau)$ depends on $l_i(\tau), l_j(\tau)$, and on the physical dimensions of the tanks, and it is obtained as follows. Let A be the cross sectional area of each tank, and a be the cross sectional area of the connecting and outlet pipes. The volume balance difference equations of each tank are then the following:

$$\begin{aligned} \frac{A}{\Delta\tau} \cdot (l_1(\tau + \Delta\tau) - l_1(\tau)) &= q_1(\tau) - q_{12}(\tau) \\ \frac{A}{\Delta\tau} \cdot (l_2(\tau + \Delta\tau) - l_2(\tau)) &= q_{12}(\tau) - q_{23}(\tau) \\ \frac{A}{\Delta\tau} \cdot (l_3(\tau + \Delta\tau) - l_3(\tau)) &= q_2(\tau) + q_{23}(\tau) - q_3(\tau). \end{aligned} \quad (3.6)$$

We then apply Torricelli's law to the equations in (3.6) to obtain the flow rates q_{12} and q_{23} :

$$\begin{aligned} q_{12}(\tau) &= \begin{cases} a_{12}(\tau) \cdot a \cdot \sqrt{2g(l_1(\tau) - l_2(\tau))} & \text{if } l_1(\tau) \geq l_2(\tau) \\ -a_{12}(\tau) \cdot a \cdot \sqrt{2g(l_2(\tau) - l_1(\tau))} & \text{otherwise;} \end{cases} \\ q_{23}(\tau) &= \begin{cases} a_{23}(\tau) \cdot a \cdot \sqrt{2g(l_2(\tau) - l_3(\tau))} & \text{if } l_2(\tau) \geq l_3(\tau) \\ -a_{23}(\tau) \cdot a \cdot \sqrt{2g(l_3(\tau) - l_2(\tau))} & \text{otherwise;} \end{cases} \end{aligned} \quad (3.7)$$

where g is the gravitational constant, and a_{12}, a_{23} are the loss coefficients of the respective pipes. These coefficients are represented as time dependent functions since they depend on the geometry of the pipes and on the water level in the tanks. In our experiments, we use the approximation $g = 9.81$, and the following values for the aforementioned coefficients: (i) $a = 0.5$, (ii) $a_{12} = 0.75$, (iii) $a_{23} = 0.75$. ◀

We now proceed to formally introduce the three components of our systems and their interaction.

3.1. Modelling the data space. We define the data space by means of a *finite* set of *variables* Var representing: (i) *environmental conditions*, such as pressure, temperature, humidity, etc., (ii) *values perceived by sensors*, which depend on the value of environmental conditions and are unavoidably affected by imprecision and approximations introduced by sensors, and (iii) *state of actuators*, which are usually elements in a discrete domain, like $\{\text{on}, \text{off}\}$ (which can be easily mapped to reals as, e.g., $\{0, 1\}$). For each $x \in \text{Var}$ we assume a measurable space $(\mathcal{D}_x, \mathcal{B}_x)$, with $\mathcal{D}_x \subseteq \mathbb{R}$ the domain of x and \mathcal{B}_x the Borel σ -algebra on \mathcal{D}_x . Without losing generality, we can assume that each domain \mathcal{D}_x is either a *finite set* or a *compact* subset of \mathbb{R} . In particular, this means that each \mathcal{D}_x is a Polish space.

As Var is a finite set, we can always assume it to be ordered, namely $\text{Var} = \{x_1, \dots, x_n\}$ for a suitable $n \in \mathbb{N}$.

Definition 3.3 (Data space). We define the *data space* over Var , notation \mathcal{D}_{Var} , as the Cartesian product of the variables domains, namely $\mathcal{D}_{\text{Var}} = \times_{i=1}^n \mathcal{D}_{x_i}$. Then, as a σ -algebra on \mathcal{D}_{Var} we consider the the product σ -algebra $\mathcal{B}_{\mathcal{D}_{\text{Var}}} = \otimes_{i=1}^n \mathcal{B}_{x_i}$.

When no confusion arises, we will use \mathcal{D} and $\mathcal{B}_{\mathcal{D}}$ in place of \mathcal{D}_{Var} and $\mathcal{B}_{\mathcal{D}_{\text{Var}}}$, respectively.

Example 3.4. The data space \mathcal{D}_{3T} for the considered three-tanks system is defined on the set of variables $\text{Var} = \{q_1, q_2, q_3, l_1, l_2, l_3\}$, with domains $q_i \in \mathcal{D}_q = [0, q_{max}]$ and $l_i \in \mathcal{D}_l = [l_{min}, l_{max}]$, for $i = 1, 2, 3$. We remark that Δ_l is not included in the data space. This is due to the fact that, in our setting, Δ_l is never modified by the environment (it is also constant in time), and it is an inherent property of the program: different programs might have different values of Δ_l , according to how capable they are of controlling and modifying the flow rates, or of reading the levels of water in the tanks. \blacktriangleleft

Elements in \mathcal{D} are n -ples of the form (v_1, \dots, v_n) , with $v_i \in \mathcal{D}_{x_i}$, which can be also identified by means of functions $\mathbf{d}: \text{Var} \rightarrow \mathbb{R}$ from variables to values, with $\mathbf{d}(x) \in \mathcal{D}_x$ for all $x \in \text{Var}$. Each function \mathbf{d} identifies a particular configuration of the data in the data space, and it is thus called a *data state*.

Definition 3.5 (Data state). A *data state* is a mapping $\mathbf{d}: \text{Var} \rightarrow \mathbb{R}$ from state variables to values, with $\mathbf{d}(x) \in \mathcal{D}_x$ for all $x \in \text{Var}$.

For simplicity, we shall write $\mathbf{d} \in \mathcal{D}$ in place of $(\mathbf{d}(x_1), \dots, \mathbf{d}(x_n)) \in \mathcal{D}$. Since program and environment interact on the basis of the *current* values of data, we have that at each step there is a data state \mathbf{d} that identifies the *current state of the data space* on which the next computation step is built. Given a data state \mathbf{d} , we let $\mathbf{d}[x = v]$ denote the data state \mathbf{d}' associating v with x , and $\mathbf{d}(y)$ with y for any $y \neq x$. For $\bar{x} = (x_{i_1}, \dots, x_{i_k})$ and $\bar{v} = (v_{i_1}, \dots, v_{i_k})$, we let $\mathbf{d}[\bar{x} = \bar{v}]$ denote $\mathbf{d}[x_{i_1} = v_{i_1}] \cdots [x_{i_k} = v_{i_k}]$.

3.2. Modelling processes. We introduce a simple process calculus allowing us to specify programs that interact with a data state \mathbf{d} in a given environment. We assume that the action performed by a process at a given computation step is determined probabilistically, according to the *generative* probabilistic model [vGSS95].

Definition 3.6 (Syntax of processes). We let \mathcal{P} be the set of *processes* P defined by:

$$P ::= (\bar{e} \rightarrow \bar{x}).P' \mid \text{if } [e] P_1 \text{ else } P_2 \mid \sum_{i \in I} p_i \cdot P_i \mid P_1 \parallel_p P_2 \mid P_1 \mid P_2 \mid A$$

$$e ::= v \in \mathbb{R} \mid x \in \text{Var} \mid \text{op}_k(e_1, \dots, e_k)$$

where p, p_1, \dots range over *probability weights* in $[0, 1]$, I is finite, A ranges over *process variables*, op_k indicates a *measurable operator* $\mathbb{R}^k \rightarrow \mathbb{R}$, and $\bar{\cdot}$ denotes a finite sequence of elements. We assume that for each process variable A we have a single definition $A \stackrel{\text{def}}{=} P$. Moreover, we require that $\sum_{i \in I} p_i = 1$ for any process $\sum_{i \in I} p_i \cdot P_i$, and that $\text{var}(P_1) \cap \text{var}(P_2) = \emptyset$ for any process $P_1 \mid P_2$, where $\text{var}(P) = \{x \in \text{Var} \mid x \in \bar{x} \text{ for any } (\bar{e} \rightarrow \bar{x}) \text{ occurring in } P\}$.

As usual, the expression $(\bar{e} \rightarrow \bar{x})$ in $(\bar{e} \rightarrow \bar{x}).P'$ is called a *prefix*. In a single action a process can read and update a set of state variables. This is done by process $(\bar{e} \rightarrow \bar{x}).P$ that first evaluates the sequence of expressions \bar{e} in the current data state and then assigns the results to sequence of variables \bar{x} , which are stored in the data state and evolve according to

| | |
|-------|---|
| (PR1) | $\text{pstep}((\bar{e} \rightarrow \bar{x}).P', \mathbf{d}) = \delta_{(\bar{x} \leftarrow \llbracket \bar{e} \rrbracket_{\mathbf{d}}, P')}$ |
| (PR2) | $\text{pstep}(\text{if } [e] P_1 \text{ else } P_2, \mathbf{d}) = \begin{cases} \text{pstep}(P_1, \mathbf{d}) & \text{if } \llbracket e \rrbracket_{\mathbf{d}} = 1 \\ \text{pstep}(P_2, \mathbf{d}) & \text{if } \llbracket e \rrbracket_{\mathbf{d}} = 0 \end{cases}$ |
| (PR3) | $\text{pstep}(\sum_i p_i \cdot P_i, \mathbf{d}) = \sum_i p_i \cdot \text{pstep}(P_i, \mathbf{d})$ |
| (PR4) | $\text{pstep}(P_1 \parallel_p P_2, \mathbf{d}) = p \cdot (\text{pstep}(P_1, \mathbf{d}) \parallel_p P_2) + (1 - p) \cdot (P_1 \parallel_p \text{pstep}(P_2, \mathbf{d}))$ |
| (PR5) | $\text{pstep}(P_1 \mid P_2, \mathbf{d}) = \text{pstep}(P_1, \mathbf{d}) \mid \text{pstep}(P_2, \mathbf{d})$ |
| (PR6) | $\text{pstep}(A, \mathbf{d}) = \text{pstep}(P, \mathbf{d}) \quad (\text{if } A \stackrel{\text{def}}{=} P)$ |

TABLE 1. Process Semantics

the environment evolution \mathcal{E} (Definition 3.9 below) before being available at next step, when the process will behave as P . We use $\sqrt{}$ to denote the empty prefix, i.e., the prefix in which the sequences \bar{e} and \bar{x} are empty, meaning that, in the current time step, the process does not read or update variables. Process $\text{if } [e] P_1 \text{ else } P_2$ behaves either as P_1 , if e evaluates to 1, or as P_2 , if e evaluates to 0. Then, $\sum_{i \in I} p_i \cdot P_i$ is the *generative probabilistic choice*: it has probability p_i to behave as P_i . We may write $\sum_{i=1}^n p_i \cdot P_i$ as $p_1 \cdot P_1 + \dots + p_n \cdot P_n$. The *generative probabilistic interleaving* construct $P_1 \parallel_p P_2$ lets the two argument processes to interleave their actions, where at each step the first process moves with probability p and the second with probability $(1 - p)$. We also provide a *synchronous parallel composition* construct $P_1 \mid P_2$ that lets the two arguments perform their actions in a synchronous fashion, provided P_1 and P_2 do not modify the same variables. Finally, process variable A allows us to specify recursive behaviours by equations of the form $A \stackrel{\text{def}}{=} P$. To avoid Zeno behaviours we assume that all occurrences of process variables appear *guarded* by prefixing constructs in P . We assume the standard notion of *free* and *bound* process variable and we only consider *closed process terms*, that is terms without *free* variables.

Actions performed by a process can be abstracted in terms of the *effects* they have on the data state, i.e., via *substitutions* of the form $\theta = [x_{i_1} \leftarrow v_{i_1}, \dots, x_{i_k} \leftarrow v_{i_k}]$, also denoted by $\bar{x} \leftarrow \bar{v}$ for $\bar{x} = x_{i_1}, \dots, x_{i_k}$ and $\bar{v} = v_{i_1}, \dots, v_{i_k}$. Since in Definition 3.6 process operations on data and expressions are assumed to be measurable, we can model the effects as $\mathcal{B}_{\mathcal{D}}$ -measurable functions $\theta: \mathcal{D} \rightarrow \mathcal{D}$ such that $\theta(\mathbf{d}) = \mathbf{d}[\bar{x} = \bar{v}]$ whenever $\theta = \bar{x} \leftarrow \bar{v}$. We denote by Θ the set of effects. The behaviour of a process can then be defined by means of a function $\text{pstep}: \mathcal{P} \times \mathcal{D} \rightarrow \Pi(\Theta \times \mathcal{P})$ that given a process P and a data state \mathbf{d} yields a *discrete* distribution over $\Theta \times \mathcal{P}$. The distributions in $\Pi(\Theta \times \mathcal{P})$ are ranged over by π, π', \dots . Function pstep is formally defined in Table 1.

In detail, rule (PR1) expresses that in a single action a process can read and update a set of variables: process $(\bar{e} \rightarrow \bar{x}).P$ first evaluates the sequence of expressions \bar{e} in the data state \mathbf{d} and then stores the results to \bar{x} . These *new* data will evolve according to \mathcal{E} before being available at next step, when the process will behave as P . The evaluation of an expression e in \mathbf{d} , denoted by $\llbracket e \rrbracket_{\mathbf{d}}$, is obtained by replacing each variable x in e with $\mathbf{d}(x)$ and by evaluating the resulting ground expression. Then, process $(\bar{e} \rightarrow \bar{x}).P$ evolves to the Dirac's distribution on the pair $(\bar{x} \leftarrow \llbracket \bar{e} \rrbracket_{\mathbf{d}}, P)$. Rule (PR2) models the behaviour of $\text{if } [e] P_1 \text{ else } P_2$. We assume that boolean operations, like those in the if-guard, evaluate to 1 or 0, standing respectively for \top and \perp . Hence, $\text{if } [e] P_1 \text{ else } P_2$ behaves as P_1 if $\llbracket e \rrbracket_{\mathbf{d}} = 1$, and it behaves as P_2 otherwise. Rule (PR3) follows the *generative* approach to probabilistic

choice: the behaviour of process P_i is selected with probability p_i . Rule (PR4) states that process $P_1 \parallel_p P_2$ interleaves the moves by P_1 and P_2 , where at each step P_1 moves with probability p and P_2 with probability $(1 - p)$. Given $\pi \in \Pi(\Theta \times \mathcal{P})$, we let $\pi \parallel_p P$ (resp. $P \parallel_p \pi$) denote the probability distribution π' over $(\Theta \times \mathcal{P})$ such that: $\pi'(\theta, P') = \pi(\theta, P'')$, whenever $P' = P'' \parallel_p P$ (resp. $P' = P \parallel_p P''$), and 0, otherwise. Rule (PR5) gives us the semantics of the *synchronous* parallel composition: for $\pi_1, \pi_2 \in \Pi(\Theta \times \mathcal{P})$, we let $\pi_1 \mid \pi_2$ denote the probability distribution defined, for all $\theta_i \in \Theta$ and $P_i \in \mathcal{P}$, $i = 1, 2$, by:

$$(\pi_1 \mid \pi_2)(\theta_1\theta_2, P_1 \mid P_2) = \pi_1(\theta_1, P_1) \cdot \pi_2(\theta_2, P_2)$$

where the effect $\theta_1\theta_2$ is given by the concatenation of the effects θ_1 and θ_2 . Notice that the concatenation is well-defined and it does not introduce contrasting effects on variables since we are assuming that $\text{var}(P_1) \cap \text{var}(P_2) = \emptyset$. Finally, rule (PR6) states that A behaves like P whenever $A \stackrel{\text{def}}{=} P$.

We can observe that pstep returns a discrete distribution over $\Theta \times \mathcal{P}$.

Proposition 3.7 (Properties of process semantics). *Let $P \in \mathcal{P}$ and $\mathbf{d} \in \mathcal{D}$. Then $\text{pstep}(P, \mathbf{d})$ is a distribution with finite support, namely the set $\text{supp}(\text{pstep}(P, \mathbf{d})) = \{(\theta, P') \mid (\text{pstep}(P, \mathbf{d}))(\theta, P') > 0\}$ is finite and $\sum_{(\theta, P') \in \text{supp}(\text{pstep}(P, \mathbf{d}))} (\text{pstep}(P, \mathbf{d}))(\theta, P') = 1$.*

Proof. The proof follows by a simple induction on the construction of $\text{pstep}(P, \mathbf{d})$. □

Example 3.8. We proceed to define the program P_{Tanks} responsible for controlling the two pumps in order to achieve (and maintain) the desired level of water in the three tanks. Intuitively, P_{Tanks} will consist of a process P_{in} controlling the flow rate through the pump attached to the first tank, and of a process P_{out} controlling the flow rate through the outlet pump. The behaviour of P_{in} and P_{out} can be obtained as a straightforward translation of, respectively, Equation (3.3) and Equation (3.4) into our simple process calculus. Formally:

$$\begin{aligned} P_{\text{in}} &\stackrel{\text{def}}{=} \begin{aligned} &\text{if } [l_1 > l_{\text{goal}} + \Delta_l] (\max\{0, q_1 - q_{\text{step}}\} \rightarrow q_1) \cdot P_{\text{in}} \\ &\text{else if } [l_1 < l_{\text{goal}} - \Delta_l] (\min\{q_{\text{max}}, q_1 + q_{\text{step}}\} \rightarrow q_1) \cdot P_{\text{in}} \\ &\text{else } \surd \cdot P_{\text{in}} \end{aligned} \\ P_{\text{out}} &\stackrel{\text{def}}{=} \begin{aligned} &\text{if } [l_3 > l_{\text{goal}} + \Delta_l] (\min\{q_{\text{max}}, q_3 + q_{\text{step}}\} \rightarrow q_3) \cdot P_{\text{out}} \\ &\text{else if } [l_3 < l_{\text{goal}} - \Delta_l] (\max\{0, q_3 - q_{\text{step}}\} \rightarrow q_3) \cdot P_{\text{out}} \\ &\text{else } \surd \cdot P_{\text{out}} \end{aligned} \end{aligned}$$

However, for P_{Tanks} to work properly, we need to allow P_{in} and P_{out} to execute simultaneously. Since the effects of read/update operators in P_{in} and P_{out} are applied to distinct variables (respectively, q_1 and q_3), we can obtain the desired execution by defining P_{Tanks} as the synchronous parallel composition of P_{in} and P_{out} :

$$P_{\text{Tanks}} \stackrel{\text{def}}{=} P_{\text{in}} \mid P_{\text{out}}$$

◀

3.3. Modelling the environment. To model the action of the environment on data we use a mapping \mathcal{E} , called *environment evolution*, taking a data state to a distribution over data states.

Definition 3.9 (Environment evolution). An *environment evolution* is a function $\mathcal{E}: \mathcal{D} \rightarrow \Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$ s.t. for each $\mathbb{D} \in \mathcal{B}_{\mathcal{D}}$ the mapping $\mathbf{d} \mapsto \mathcal{E}(\mathbf{d})(\mathbb{D})$ is $\mathcal{B}_{\mathcal{D}}$ -measurable.

We notice that, due to the interaction with the program, in our model the probability induced by \mathcal{E} at the next time step *depends only* on the current state of the data space. It is then natural to assume that the behaviour of the environment is modelled as a discrete time Markov process, and to define function \mathcal{E} we only need the initial distribution on \mathcal{D} (which in our setting will be the Dirac's distribution on a data state in \mathcal{D}) and the random vector $\mathbf{X} = [X_1, \dots, X_n]$ in which each random variable X_i models the changes induced by the environment on the value of variable x_i .

Since the environment is an ensemble of physical phenomena, it is represented by a system of equations of the form $x' = f(x)$, for $x \in \text{Var}$, and thus with the obvious syntax. Hence, for simplicity, we do not explicitly present a syntactic definition of the environment, but we model it directly as a function over data spaces. The following example shows how the environment evolution is obtained from the system of equations.

Example 3.10. To conclude the encoding of the three-tanks experiment from Example 3.2 into our model, we need to define a suitable environment evolution. This can be derived from Equations (3.5), (3.1) (or (3.2)), and (3.7) in the obvious way. For sake of completeness, below we present functions \mathcal{E}_1 and \mathcal{E}_2 modelling the evolution of the environment, respectively, in scenario 1 (with Equation (3.1)) and scenario 2 (with Equation (3.2)). We remark that:

- The only stochastic variable in our setting is q_2 , giving thus that the distribution induced by \mathcal{E}_i is determined by the distribution of q_2 , for $i = 1, 2$.
- The values of q_1 and q_3 are modified by \mathbf{P}_{in} and \mathbf{P}_{out} , respectively. In particular, according to Definition 3.13, this means that the environment does not affect the values of q_1 and q_3 and, moreover, when we evaluate $\mathcal{E}_i(\mathbf{d})$, the effects of the two process over q_1 and q_3 have already been taken into account in \mathbf{d} , for $i = 1, 2$.

Define $f_{3T}: [0, q_{max}] \times \mathcal{D}_{3T} \rightarrow \mathcal{D}_{3T}$ as the function such that, for any $x \in [0, q_{max}]$ and data state \mathbf{d} , gives us the data state $f_{3T}(x, \mathbf{d}) = \mathbf{d}'$ obtained from the solution of the following system of equations:

$$\left\{ \begin{array}{l} q_{12} = \begin{cases} a_{12} \cdot a \cdot \sqrt{2g(\mathbf{d}(l_1) - \mathbf{d}(l_2))} & \text{if } \mathbf{d}(l_1) \geq \mathbf{d}(l_2) \\ -a_{12} \cdot a \cdot \sqrt{2g(\mathbf{d}(l_2) - \mathbf{d}(l_1))} & \text{otherwise;} \end{cases} \\ q_{23} = \begin{cases} a_{23} \cdot a \cdot \sqrt{2g(\mathbf{d}(l_2) - \mathbf{d}(l_3))} & \text{if } \mathbf{d}(l_2) \geq \mathbf{d}(l_3) \\ -a_{23} \cdot a \cdot \sqrt{2g(\mathbf{d}(l_3) - \mathbf{d}(l_2))} & \text{otherwise;} \end{cases} \\ \mathbf{d}'(l_1) = \mathbf{d}(l_1) + \Delta\tau \cdot (\mathbf{d}(q_1) - q_{12}) \\ \mathbf{d}'(l_2) = \mathbf{d}(l_2) + \Delta\tau \cdot (q_{12} - q_{23}) \\ \mathbf{d}'(l_3) = \mathbf{d}(l_3) + \Delta\tau \cdot (x + q_{23} - \mathbf{d}(q_3)) \\ \mathbf{d}'(q_1) = \mathbf{d}(q_1) \\ \mathbf{d}'(q_2) = x \\ \mathbf{d}'(q_3) = \mathbf{d}(q_3) \end{array} \right.$$

where $a, a_{12}, a_{23}, g, \Delta\tau$ are constants. Then, consider the random variables $X \sim N(q_{med}, \Delta_q)$ and $Y \sim N(0, 1)$. From them, we define the random variables $Q_2^1(\mathbf{d}) = f_{3T}(X, \mathbf{d})$ and $Q_2^2(\mathbf{d}) = f_{3T}(\max\{q_{max}, \mathbf{d}(q_2) + Y\}, \mathbf{d})$. Then, for $i = 1, 2$ and for each data state \mathbf{d} , the distribution $\mathcal{E}_i(\mathbf{d})$ is obtained as the distribution of Q_2^i . \blacktriangleleft

3.4. Modelling system's behaviour. We use the notion of *configuration* to model the state of the system at each time step.

Definition 3.11 (Configuration). A *configuration* is a triple $c = \langle P, \mathbf{d} \rangle_{\mathcal{E}}$, where P is a process, \mathbf{d} is a data state and \mathcal{E} is an environment evolution. We denote by $\mathcal{C}_{\mathcal{P}, \mathcal{D}, \mathcal{E}}$ the set of configurations defined over \mathcal{P}, \mathcal{D} and \mathcal{E} .

When no confusion shall arise, we shall write \mathcal{C} in place of $\mathcal{C}_{\mathcal{P}, \mathcal{D}, \mathcal{E}}$.

Let $(\mathcal{P}, \Sigma_{\mathcal{P}})$ be the measurable space of processes, where $\Sigma_{\mathcal{P}}$ is the power set of \mathcal{P} , and $(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$ be the measurable space of data states. As \mathcal{E} is fixed, we can identify \mathcal{C} with $\mathcal{P} \times \mathcal{D}$ and equip it with the product σ -algebra $\Sigma_{\mathcal{C}} = \Sigma_{\mathcal{P}} \otimes \mathcal{B}_{\mathcal{D}}$: $\Sigma_{\mathcal{C}}$ is generated by the sets $\{\langle \mathbb{P}, \mathbb{D} \rangle_{\mathcal{E}} \mid \mathbb{P} \in \Sigma_{\mathcal{P}}, \mathbb{D} \in \mathcal{B}_{\mathcal{D}}\}$, where $\langle \mathbb{P}, \mathbb{D} \rangle_{\mathcal{E}} = \{\langle P, \mathbf{d} \rangle_{\mathcal{E}} \mid P \in \mathbb{P}, \mathbf{d} \in \mathbb{D}\}$.

Notation 3.12. For $\mu_{\mathcal{P}} \in \Pi(\mathcal{P}, \Sigma_{\mathcal{P}})$ and $\mu_{\mathcal{D}} \in \Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$ we let $\mu = \langle \mu_{\mathcal{P}}, \mu_{\mathcal{D}} \rangle_{\mathcal{E}}$ denote the product distribution on $(\mathcal{C}, \Sigma_{\mathcal{C}})$, i.e., $\mu(\langle \mathbb{P}, \mathbb{D} \rangle_{\mathcal{E}}) = \mu_{\mathcal{P}}(\mathbb{P}) \cdot \mu_{\mathcal{D}}(\mathbb{D})$ for all $\mathbb{P} \in \Sigma_{\mathcal{P}}$ and $\mathbb{D} \in \mathcal{B}_{\mathcal{D}}$. If $\mu_{\mathcal{P}} = \delta_P$ for some $P \in \mathcal{P}$, we shall denote $\langle \delta_P, \mu_{\mathcal{D}} \rangle_{\mathcal{E}}$ simply by $\langle P, \mu_{\mathcal{D}} \rangle_{\mathcal{E}}$.

Our aim is to express the behaviour of a system in terms of the changes on data. We start with the *one-step* behaviour of a configuration, in which we combine the effects on the data state induced by the activity of the process (given by `pstep`) and the subsequent action by the environment. Formally, we define a function `cstep` that, given a configuration c , yields a distribution on $(\mathcal{C}, \Sigma_{\mathcal{C}})$ (Definition 3.13 below). Then, we use `cstep` to define the *multi-step* behaviour of configuration c as a sequence $\mathcal{S}_{c,0}^{\mathcal{C}}, \mathcal{S}_{c,1}^{\mathcal{C}}, \dots$ of distributions on $(\mathcal{C}, \Sigma_{\mathcal{C}})$. To this end, we show that `cstep` is a Markov kernel (Proposition 3.15 below). Finally, to abstract from processes and focus only on data, from the sequence $\mathcal{S}_{c,0}^{\mathcal{C}}, \mathcal{S}_{c,1}^{\mathcal{C}}, \dots$, we obtain a sequence of distributions $\mathcal{S}_{c,0}^{\mathcal{D}}, \mathcal{S}_{c,1}^{\mathcal{D}}, \dots$ on $(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$ called the *evolution sequence* of the system (Definition 3.18 below).

Definition 3.13 (One-step semantics of configurations). Function `cstep`: $\mathcal{C} \rightarrow \Pi(\mathcal{C}, \Sigma_{\mathcal{C}})$ is defined for all configurations $\langle P, \mathbf{d} \rangle_{\mathcal{E}} \in \mathcal{C}$ by

$$\text{cstep}(\langle P, \mathbf{d} \rangle_{\mathcal{E}}) = \sum_{(\theta, P') \in \text{supp}(\text{pstep}(P, \mathbf{d}))} \text{pstep}(P, \mathbf{d})(\theta, P') \cdot \langle P', \mathcal{E}(\theta(\mathbf{d})) \rangle_{\mathcal{E}}. \quad (3.8)$$

The next result follows by $\mathcal{E}(\theta(\mathbf{d})) \in \Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$ and Proposition 3.7.

Proposition 3.14 (Properties of one-step configuration semantics). *Assume any configuration $\langle P, \mathbf{d} \rangle_{\mathcal{E}} \in \mathcal{C}$. Then `cstep`($\langle P, \mathbf{d} \rangle_{\mathcal{E}}$) is a distribution on $(\mathcal{C}, \Sigma_{\mathcal{C}})$.*

Proof. Being \mathbf{d} a data state in \mathcal{D} and θ an effect, $\theta(\mathbf{d})$ is a data state. Then, $\mathcal{E}(\theta(\mathbf{d}))$ is a distribution on $(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$, thus implying that $\langle P', \mathcal{E}(\theta(\mathbf{d})) \rangle_{\mathcal{E}}$ is a distribution on $(\mathcal{C}, \Sigma_{\mathcal{C}})$ (see Notation 3.12). Finally, by Proposition 3.7 we get that `cstep`($\langle P, \mathbf{d} \rangle_{\mathcal{E}}$) is a convex combination of distributions on $(\mathcal{C}, \Sigma_{\mathcal{C}})$ whose weights sum up to 1, which gives the thesis. \square

Since $\text{cstep}(c) \in \Pi(\mathcal{C}, \Sigma_{\mathcal{C}})$ for each $c \in \mathcal{C}$, we can rewrite $\text{cstep}: \mathcal{C} \times \Sigma_{\mathcal{C}} \rightarrow [0, 1]$, so that for each configuration $c \in \mathcal{C}$ and measurable set $\mathbb{C} \in \Sigma_{\mathcal{C}}$, $\text{cstep}(c)(\mathbb{C})$ denotes the probability of reaching in one step a configuration in \mathbb{C} starting from c . We can prove that cstep is the Markov kernel of the Markov process modelling our system. This follows by Proposition 3.14 and by proving that for each $\mathbb{C} \in \Sigma_{\mathcal{C}}$, the mapping $c \mapsto \text{cstep}(c)(\mathbb{C})$ is $\Sigma_{\mathcal{C}}$ -measurable.

Proposition 3.15. *The function cstep is a Markov kernel.*

Proof. We need to show that cstep satisfies the two properties of Markov kernels, namely

- (1) For each configuration $c \in \mathcal{C}$, the mapping $\mathbb{C} \mapsto \text{cstep}(c)(\mathbb{C})$ is a distribution on $(\mathcal{C}, \Sigma_{\mathcal{C}})$.
- (2) For each measurable set $\mathbb{C} \in \Sigma_{\mathcal{C}}$, the mapping $c \mapsto \text{cstep}(c)(\mathbb{C})$ is $\Sigma_{\mathcal{C}}$ -measurable.

Item 1 follows directly by Proposition 3.14.

Let us focus on item 2. By Definition 3.13, for each $\langle P, \mathbf{d} \rangle_{\mathcal{E}} \in \mathcal{C}$ we have that

$$\text{cstep}(\langle P, \mathbf{d} \rangle_{\mathcal{E}})(\mathbb{C}) = \sum_{(\theta, P') \in \text{supp}(\text{pstep}(P, \mathbf{d}))} \text{pstep}(P, \mathbf{d})(\theta, P') \cdot \langle P', \mathcal{E}(\theta(\mathbf{d})) \rangle_{\mathcal{E}}(\mathbb{C}) .$$

As, by definition, each $\theta \in \Theta$ and \mathcal{E} are $\mathcal{B}_{\mathcal{D}}$ -measurable functions, we can infer that also their composition $\mathcal{E}(\theta(\cdot))$ is a $\mathcal{B}_{\mathcal{D}}$ -measurable function. Since, moreover, $\Sigma_{\mathcal{C}}$ is the (smallest) σ -algebra generated by $\Sigma_{\mathcal{P}} \times \mathcal{B}_{\mathcal{D}}$ and every subset of \mathcal{P} is a measurable set in $\Sigma_{\mathcal{P}}$, we can also infer that $\langle (\cdot), \mathcal{E}(\theta(\cdot)) \rangle_{\mathcal{E}}$ is a $\Sigma_{\mathcal{C}}$ -measurable function. Finally, we recall that by Proposition 3.7 we have that $\text{supp}(\text{pstep}(P, \mathbf{d}))$ is finite. Therefore, we can conclude that cstep is a $\Sigma_{\mathcal{C}}$ -measurable function as linear combination of a finite collection of $\Sigma_{\mathcal{C}}$ -measurable functions (see, e.g., [Roy88, Chapter 3.5, Proposition 19]). \square

Hence, the multi-step behaviour of configuration c can be defined as a time homogeneous Markov process having cstep as Markov kernel and δ_c as initial distribution.

Definition 3.16 (Multi-step semantics of configurations). Let $c \in \mathcal{C}$ be a configuration. The *multi-step behaviour* of c is the sequence of distributions $\mathcal{S}_{c,0}^{\mathcal{C}}, \mathcal{S}_{c,1}^{\mathcal{C}}, \dots$ on $(\mathcal{C}, \Sigma_{\mathcal{C}})$ defined inductively as follows:

$$\begin{aligned} \mathcal{S}_{c,0}^{\mathcal{C}}(\mathbb{C}) &= \delta_c(\mathbb{C}), \text{ for all } \mathbb{C} \in \Sigma_{\mathcal{C}} \\ \mathcal{S}_{c,i+1}^{\mathcal{C}}(\mathbb{C}) &= \int_{\mathcal{C}} \text{cstep}(b)(\mathbb{C}) \, d(\mathcal{S}_{c,i}^{\mathcal{C}}(b)), \text{ for all } \mathbb{C} \in \Sigma_{\mathcal{C}}. \end{aligned}$$

We can prove that $\mathcal{S}_{c,0}^{\mathcal{C}}, \mathcal{S}_{c,1}^{\mathcal{C}}, \dots$ are well defined, namely they are distributions on $(\mathcal{C}, \Sigma_{\mathcal{C}})$.

Proposition 3.17 (Properties of configuration multi-step semantics). *For any $c \in \mathcal{C}$, all $\mathcal{S}_{c,0}^{\mathcal{C}}, \mathcal{S}_{c,1}^{\mathcal{C}}, \dots$ are distributions on $(\mathcal{C}, \Sigma_{\mathcal{C}})$.*

Proof. The proof follows by induction. The base case for $\mathcal{S}_{c,0}^{\mathcal{C}}$ is immediate. The inductive step follows by Proposition 3.15. \square

As the program-environment interplay can be observed only in the changes they induce on the data states, we define the *evolution sequence* of a configuration as the sequence of distributions over data states that are reached by it, step-by-step.

Definition 3.18 (Evolution sequence). The *evolution sequence* of a configuration $c = \langle P, \mathbf{d} \rangle_{\mathcal{E}}$ is a sequence $\mathcal{S}_c^{\mathcal{D}} \in \Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}})^{\omega}$ of distributions on $(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$ s.t. $\mathcal{S}_c^{\mathcal{D}} = \mathcal{S}_{c,0}^{\mathcal{D}} \dots \mathcal{S}_{c,n}^{\mathcal{D}} \dots$ if and only if for all $i \geq 0$ and for all $\mathbb{D} \in \mathcal{B}_{\mathcal{D}}$,

$$\mathcal{S}_{c,i}^{\mathcal{D}}(\mathbb{D}) = \mathcal{S}_{c,i}^{\mathcal{C}}(\langle \mathcal{P}, \mathbb{D} \rangle_{\mathcal{E}}).$$

4. TOWARDS A METRIC FOR SYSTEMS

As outlined in the Introduction, we aim at defining a *distance* over the systems described in the previous section, called the *evolution metric*, allowing us to do the following:

- (1) Measure how well a program is fulfilling its tasks.
- (2) Establish whether one program behaves better than another one in an environment.
- (3) Compare the interactions of a program with different environments.

As we will see, these three objectives can be naturally obtained thanks to the possibility of modelling the program in isolation from the environment typical of our model, and to our purely data-driven system semantics. Intuitively, since the behaviour of a system is entirely described by its evolution sequence, the evolution metric \mathbf{m} will indeed be defined as a distance on the evolution sequences of systems. However, in order to obtain the proper technical definition of \mathbf{m} , some considerations are due.

Firstly, we notice that in most applications, not only the program-environment interplay, but also the tasks of the program can be expressed in a purely data-driven fashion. For instance, if we are monitoring a particular feature of the system then we can identify a (time-dependent) *specific value of interest* of a state variable, like an upper bound on the energy consumption at a given time step. Similarly, if we aim at controlling several interacting features or there is a range of parameters that can be considered acceptable, like in the case of our running example, then we can specify a (time-dependent) *set of values of interest*. At any time step, any difference between them and the data actually obtained can be interpreted as a flaw in systems behaviour. We use a *penalty function* ρ to quantify these differences. From the penalty function we can obtain a *distance on data states*, namely a 1-bounded *hemimetric* $m^{\mathcal{D}}$ expressing how much a data state \mathbf{d}_2 is worse than a data state \mathbf{d}_1 according to parameters of interests. The idea is that if \mathbf{d}_1 is obtained by the interaction of P_1 with \mathcal{E} and \mathbf{d}_2 is obtained from the one of P_2 with \mathcal{E} , \mathbf{d}_2 being worse than \mathbf{d}_1 means that P_2 is performing worse than P_1 in this case. Similarly, if \mathbf{d}_1 and \mathbf{d}_2 are obtained from the interaction of P with \mathcal{E}_1 and \mathcal{E}_2 , respectively, we can express whether in this particular configuration and time step P behaves worse in one environment with respect to the other.

Secondly, we recall that the evolution sequence of a system consists in a sequence of *distributions* over data states. Hence, we use the *Wasserstein metric* to lift $m^{\mathcal{D}}$ to a distance $\mathbf{W}(m^{\mathcal{D}})$ over distributions over data states. Informally, the Wasserstein metric gives the expected value of the ground distance between the elements in the support of the distributions. Thus, in our case, the lifting expresses how much worse a configuration is expected to behave with respect to another one at a given time.

Finally, we need to lift $\mathbf{W}(m^{\mathcal{D}})$ to a distance on the entire evolution sequences of systems. For our purposes, a reasonable choice is to take the maximum over time of the pointwise (with respect to time) Wasserstein distances (see Remark 4.7 below for further details on this choice). Actually, to favour computational tractability, our metric will *not* be evaluated on *all* the distributions in the evolution sequences, but *only* on those that are reached at certain time steps, called the *observation times* (OT).

We devote the remaining of this section to a formalisation of the intuitions above.

4.1. A metric on data states. We start by proposing a metric on data states, seen as *static components* in isolation from processes and environment. To this end, we introduce a *penalty function* $\rho: \mathcal{D} \rightarrow [0, 1]$, a continuous function that assigns to each data state \mathbf{d} a penalty in $[0, 1]$ expressing how far the values of the parameters of interest in \mathbf{d} are from

their desired ones (hence $\rho(\mathbf{d}) = 0$ if \mathbf{d} respects all the parameters). Since sometimes the parameters can be time-dependent, we also introduce a time-dependent version of ρ : at any time step τ , the τ -penalty function ρ_τ compares the data states with respect to the values of the parameters expected at time τ . When the value of the penalty function is independent from the time step, we simply omit the subscript τ .

Example 4.1. A requirement on the behaviour of the three-tanks system from Example 3.2 can be that the level of water l_i should be at the level l_{goal} , for $i = 1, 2, 3$. This can be easily rendered in terms of the penalty functions ρ^{l_i} , for $i = 1, 2, 3$, defined as the normalised distance between the current level of water $\mathbf{d}(l_i)$ and l_{goal} , namely:

$$\rho^{l_i}(\mathbf{d}) = \frac{|\mathbf{d}(l_i) - l_{goal}|}{\max\{l_{max} - l_{goal}, l_{goal} - l_{min}\}} . \quad (4.1)$$

Clearly, we can also consider as a requirement that *all* the l_i are at level l_{goal} . The penalty function thus becomes

$$\rho^{\max}(\mathbf{d}) = \max_{i=1,2,3} \rho^{l_i}(\mathbf{d}) .$$

The ones proposed above are just a few simple examples of requirements, and related penalty functions, for the three-tanks experiment. Yet, in their simplicity, they will allow us to showcase a number of different applications of our framework while remaining in a setting where the reader can easily foresee the expected results. ◀

A formal definition of the penalty function is beyond the purpose of this paper, also due to its context-dependent nature. Besides, notice that we can assume that ρ already includes some tolerances with respect to the exact values of the parameters in its evaluation, and thus we do not consider them. The (*timed*) *metric on data states* is then defined as the asymmetric difference between the penalties assigned to them by the penalty function.

Definition 4.2 (Metric on data states). For any time step τ , let $\rho_\tau: \mathcal{D} \rightarrow [0, 1]$ be the τ -penalty function on \mathcal{D} . The τ -metric on data states in \mathcal{D} , $m_{\rho,\tau}^{\mathcal{D}}: \mathcal{D} \times \mathcal{D} \rightarrow [0, 1]$, is defined, for all $\mathbf{d}_1, \mathbf{d}_2 \in \mathcal{D}$, by

$$m_{\rho,\tau}^{\mathcal{D}}(\mathbf{d}_1, \mathbf{d}_2) = \max\{\rho_\tau(\mathbf{d}_2) - \rho_\tau(\mathbf{d}_1), 0\}.$$

Notice that $m_{\rho,\tau}^{\mathcal{D}}(\mathbf{d}_1, \mathbf{d}_2) > 0$ iff $\rho_\tau(\mathbf{d}_2) > \rho_\tau(\mathbf{d}_1)$, i.e., the penalty assigned to \mathbf{d}_2 is higher than that assigned to \mathbf{d}_1 . For this reason, we say that $m_{\rho,\tau}^{\mathcal{D}}(\mathbf{d}_1, \mathbf{d}_2)$ expresses *how worse* \mathbf{d}_2 is than \mathbf{d}_1 with respect to the objectives of the system. It is not hard to see that for all $\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3 \in \mathcal{D}$ we have $m_{\rho,\tau}^{\mathcal{D}}(\mathbf{d}_1, \mathbf{d}_2) \leq 1$, $m_{\rho,\tau}^{\mathcal{D}}(\mathbf{d}_1, \mathbf{d}_1) = 0$, and $m_{\rho,\tau}^{\mathcal{D}}(\mathbf{d}_1, \mathbf{d}_2) \leq m_{\rho,\tau}^{\mathcal{D}}(\mathbf{d}_1, \mathbf{d}_3) + m_{\rho,\tau}^{\mathcal{D}}(\mathbf{d}_3, \mathbf{d}_2)$, thus ensuring that $m_{\rho,\tau}^{\mathcal{D}}$ is a 1-bounded hemimetric.

Proposition 4.3. *Function $m_{\rho,\tau}^{\mathcal{D}}$ is a 1-bounded hemimetric on \mathcal{D} .*

When no confusion shall arise, we shall drop the ρ, τ subscripts. We remark that penalty functions allow us to define the distance between two data states, which are elements of \mathbb{R}^n , in terms of a distance on \mathbb{R} . As we discuss in Section 5, this feature significantly lowers the complexity of the evaluation the evolution metric.

4.2. Lifting $m^{\mathcal{D}}$ to distributions. The second step to obtain the evolution metric consists in lifting $m^{\mathcal{D}}$ to a metric on distributions on data states. In the literature, we can find a wealth of notions of distances on distributions (see [RKSF13] for a survey). For our purposes, the most suitable one is the *Wasserstein metric* [Vas69].

According to Definition 2.2 (given in Section 2), for any two distributions μ, ν on $(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$, the Wasserstein lifting of $m^{\mathcal{D}}$ to a distance between μ and ν is defined by

$$\mathbf{W}(m^{\mathcal{D}})(\mu, \nu) = \inf_{\mathfrak{w} \in \mathfrak{W}(\mu, \nu)} \int_{\mathcal{D} \times \mathcal{D}} m^{\mathcal{D}}(\mathbf{d}, \mathbf{d}') \, d\mathfrak{w}(\mathbf{d}, \mathbf{d}')$$

where $\mathfrak{W}(\mu, \nu)$ is the set of the *couplings* of μ and ν , namely the set of joint distributions \mathfrak{w} over the product space $(\mathcal{D} \times \mathcal{D}, \mathcal{B}_{\mathcal{D}} \otimes \mathcal{B}_{\mathcal{D}})$ having μ and ν as left and right marginal.

As outlined in Section 2, [FR18, Proposition 4, Proposition 6] guarantee that $\mathbf{W}(m^{\mathcal{D}})$ is a well-defined hemimetric on $\Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$.

Proposition 4.4. *Function $\mathbf{W}(m_{\rho, \tau}^{\mathcal{D}})$ is a 1-bounded hemimetric on $\Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$.*

4.3. The evolution metric. We now need to lift $\mathbf{W}(m^{\mathcal{D}})$ to a distance on evolution sequences. To this end, we observe that the evolution sequence of a configuration includes the distributions over data states induced after *each* computation step. However, it could be the case that the changes on data induced by the environment can be appreciated only along wider time intervals. To deal with this kind of situations, we introduce the notion of *observation times*, namely a *discrete* set OT of time steps at which the modifications induced by the program-environment interplay give us useful information on the evolution of the system (with respect to the considered objectives). Hence, a comparison of the evolution sequences based on the differences in the distributions reached at the times in OT can be considered meaningful. Moreover, considering only the differences at the observation times will favour the computational tractability of the evolution metric.

To compare evolution sequences, we propose a sort of *weighted infinity norm* of the tuple of the Wasserstein distances between the distributions in them. As weight we consider a non-increasing function $\lambda: \text{OT} \rightarrow (0, 1]$ allowing us to express how much the distance at time τ , namely $\mathbf{W}(m_{\rho, \tau}^{\mathcal{D}})(\mathcal{S}_{c_1, \tau}^{\mathcal{D}}, \mathcal{S}_{c_2, \tau}^{\mathcal{D}})$, affects the overall distance between configurations c_1 and c_2 . The idea is that, in certain application contexts, the differences in the behaviour of systems that are detected after wide time intervals, can be less relevant than the differences in the initial behaviour. Hence, we can use the weight $\lambda(\tau)$ to *mitigate* the distance of events at time τ . Following the terminology introduced in the context of behavioural metrics [dAHM03, DGJP04], we refer to λ as to the *discount function*, and to $\lambda(\tau)$ as to the *discount factor at time τ* . Clearly, the constant function $\lambda(\tau) = 1$, for all $\tau \in \text{OT}$, means that no discount is applied.

Definition 4.5 (Evolution metric). Assume a set OT of observation times and a discount function λ . For each $\tau \in \text{OT}$, let ρ_{τ} be a τ -penalty function and let $m_{\rho, \tau}^{\mathcal{D}}$ be the τ -metric on data states defined on it. Then, the λ -*evolution metric* over ρ and OT is the mapping $\mathfrak{m}_{\rho, \text{OT}}^{\lambda}: \mathcal{C} \times \mathcal{C} \rightarrow [0, 1]$ defined, for all configurations $c_1, c_2 \in \mathcal{C}$, by

$$\mathfrak{m}_{\rho, \text{OT}}^{\lambda}(c_1, c_2) = \sup_{\tau \in \text{OT}} \lambda(\tau) \cdot \mathbf{W}(m_{\rho, \tau}^{\mathcal{D}})(\mathcal{S}_{c_1, \tau}^{\mathcal{D}}, \mathcal{S}_{c_2, \tau}^{\mathcal{D}}).$$

Since $\mathbf{W}(m_{\rho, \tau}^{\mathcal{D}})$ is a 1-bounded hemimetric (Proposition 4.4) we can easily derive the same property for $\mathfrak{m}_{\rho, \text{OT}}^{\lambda}$.

Proposition 4.6. *Function $\mathbf{m}_{\rho, \text{OT}}^\lambda$ is a 1-bounded hemimetric on \mathcal{C} .*

Notice that if λ is a *strictly* non-increasing function, then to obtain upper bounds on the evolution metric only a *finite* number of observations is needed.

Remark 4.7. Usually, due to the presence of uncertainties, the behaviour of a system can be considered acceptable even if it differs from its intended one *up-to a certain tolerance*. Similarly, the properties of adaptability, reliability, and robustness that we aim to study will check whether a program is able to perform well in a perturbed environment *up-to a given tolerance*. In this setting, the choice of defining the evolution metric as the pointwise maximal distance in time between the evolution sequences of systems is natural and reasonable: if in the worst case (the maximal distance) the program keeps the parameters of interest within the given tolerance, then its entire behaviour can be considered acceptable. However, with this approach we have that a program is only as good as its worst performance, and one could argue that there are application contexts in which our evolution metric would be less meaningful. For these reasons, we remark that we could have given a *parametric* version of Definition 4.5 and defining the evolution metric in terms of a generic *aggregation function* f over the tuple of Wasserstein distances. Then, one could choose the best instance for f according to the chosen application context. The use of a parametric definition would have not affected the technical development of our paper. However, in order to keep the notation and presentation as simple as possible, we opted to define $\mathbf{m}_{\rho, \text{OT}}^\lambda$ directly in the weighted infinity norm form. A similar reasoning applies to the definition of the penalty function that we gave in Example 4.1, and to the definition of the metric over data states (Definition 4.2).

5. ESTIMATING THE EVOLUTION METRIC

In this section we show how the evolution metric can be estimated via statistical techniques. Firstly, in Section 5.1 we show how we can estimate the evolution sequence of a given configuration c . Then, in Section 5.2 we evaluate the distance between two configurations c_1 and c_2 on their estimated evolution sequences.

5.1. Computing empirical evolution sequences. Given a configuration $\langle P, \mathbf{d} \rangle_{\mathcal{E}}$ and an integer k we can use the function SIMULATE, defined in Figure 2, to sample a sequence of configurations of the form

$$\langle P_0, \mathbf{d}_0 \rangle_{\mathcal{E}}, \langle P_1, \mathbf{d}_1 \rangle_{\mathcal{E}}, \dots, \langle P_k, \mathbf{d}_k \rangle_{\mathcal{E}}.$$

This sequence represents k -steps of a computation starting from $\langle P, \mathbf{d} \rangle_{\mathcal{E}} = \langle P_0, \mathbf{d}_0 \rangle_{\mathcal{E}}$. Each step of the sequence is computed by using function SIMSTEP, also defined in Figure 2. There we let RAND be a function that allows us to get a uniform random number in $(0, 1]$ while SAMPLE(\mathcal{E}, \mathbf{d}) is used to sample a data state in the distribution $\mathcal{E}(\mathbf{d})$. We assume that for any measurable set of data states $\mathbb{D} \in \mathcal{B}_{\mathcal{D}}$ the probability of SAMPLE(\mathcal{E}, \mathbf{d}) to be in \mathbb{D} is equal to the probability of \mathbb{D} induced by $\mathcal{E}(\mathbf{d})$, namely:

$$Pr(\text{SAMPLE}(\mathcal{E}, \mathbf{d}) \in \mathbb{D}) = \mathcal{E}(\mathbf{d})(\mathbb{D}) . \quad (5.1)$$

Example 5.1. If we consider the environment evolution from Examples 3.2 and 3.10, function SAMPLE(\mathcal{E}, \mathbf{d}) would sample a random value x for q_2 from the distribution chosen for it (according to the considered scenario), and then solve the system of equations identified by $f_{3T}(x, \mathbf{d})$. ◀

| | |
|--|---|
| <pre> 1: function SIMULATE($\langle P, \mathbf{d} \rangle_{\mathcal{E}}, k$) 2: $i \leftarrow 1$ 3: $c \leftarrow \langle P, \mathbf{d} \rangle_{\mathcal{E}}$ 4: $a \leftarrow c$ 5: while $i \leq k$ do 6: $c \leftarrow \text{SIMSTEP}(c)$ 7: $a \leftarrow a, c$ 8: $i \leftarrow i + 1$ 9: end while 10: return a 11: end function </pre> | <pre> 1: function SIMSTEP($\langle P, \mathbf{d} \rangle_{\mathcal{E}}$) 2: $\sum_{i=1}^n p_i(\theta_i, P_i) \leftarrow \text{pstep}(P, \mathbf{d})$ 3: $u \leftarrow \text{RAND}()$ 4: let i s.t. $\sum_{j=1}^{i-1} p_j < u \leq \sum_{j=1}^i p_j$ 5: $\mathbf{d}'_i \leftarrow \text{SAMPLE}(\mathcal{E}, \theta_i(\mathbf{d}_i))$ 6: return $\langle P_i, \mathbf{d}'_i \rangle_{\mathcal{E}}$ 7: end function </pre> |
|--|---|

Figure 2. Functions used to simulate behaviour of a configuration.

We can then extend this property to configurations and function SIMSTEP:

Lemma 5.2. *For any configuration $\langle P, \mathbf{d} \rangle_{\mathcal{E}} \in \mathcal{C}$, and for any measurable set $\mathbb{C} \in \Sigma_{\mathcal{C}}$:*

$$Pr(\text{SIMSTEP}(\langle P, \mathbf{d} \rangle_{\mathcal{E}}) \in \mathbb{C}) = \text{cstep}(\langle P, \mathbf{d} \rangle_{\mathcal{E}})(\mathbb{C}).$$

Proof. To prove the thesis it is enough to show that the property

$$Pr(\text{SIMSTEP}(\langle P, \mathbf{d} \rangle_{\mathcal{E}}) \in \mathbb{C}) = \text{cstep}(\langle P, \mathbf{d} \rangle_{\mathcal{E}})(\mathbb{C})$$

holds on the generators of the σ -algebra $\Sigma_{\mathcal{C}}$. Hence, assume that $\mathbb{C} = \langle \mathbb{P}, \mathbb{D} \rangle_{\mathcal{E}}$ for some $\mathbb{P} \in \Sigma_{\mathcal{P}}$ and $\mathbb{D} \in \mathcal{B}_{\mathcal{D}}$. Then we have

$$\begin{aligned}
\text{cstep}(\langle P, \mathbf{d} \rangle_{\mathcal{E}})(\mathbb{C}) &= \sum_{(\theta, P') \in \text{supp}(\text{pstep}(P, \mathbf{d}))} (\text{pstep}(P, \mathbf{d})(\theta, P') \cdot \langle P', \mathcal{E}(\theta(\mathbf{d})) \rangle_{\mathcal{E}})(\mathbb{C}) \\
&= \sum_{i=1}^n (p_i \cdot \langle P_i, \mathcal{E}(\theta_i(\mathbf{d})) \rangle_{\mathcal{E}})(\mathbb{C}) \\
&= \sum_{i=1}^n p_i \cdot \mathbb{1}_{\mathbb{P}}\{P_i\} \cdot \mathcal{E}(\theta_i(\mathbf{d}))(\mathbb{D}) \\
&= \sum_{\{i|P_i \in \mathbb{P}\}} p_i \cdot \mathcal{E}(\theta_i(\mathbf{d}))(\mathbb{D}) \\
&= \sum_{\{i|P_i \in \mathbb{P}\}} p_i \cdot Pr(\text{SAMPLE}(\mathcal{E}, \theta_i(\mathbf{d})) \in \mathbb{D}) \\
&= Pr(\text{SIMSTEP}(\langle P, \mathbf{d} \rangle_{\mathcal{E}}) \in \mathbb{C})
\end{aligned}$$

where:

- The first step follows by the definition of cstep (Definition 3.13).
- The second step follows by $\text{pstep}(P, \mathbf{d}) = p_1 \cdot (\theta_1, P_1) + \dots + p_n \cdot (\theta_n, P_n)$, for some p_i, θ_i and P_i (see Proposition 3.7).
- The third and fourth steps follow by the definition of the distribution $\langle \mu_{\mathcal{P}}, \mu_{\mathcal{D}} \rangle_{\mathcal{E}}$ (Notation 3.12), in which $\mathbb{1}_{\mathbb{P}}$ denotes the characteristic function of the set \mathbb{P} , i.e., $\mathbb{1}_{\mathbb{P}}\{P'\} = 1$ if $P' \in \mathbb{P}$ and $\mathbb{1}_{\mathbb{P}}\{P'\} = 0$ otherwise.
- The fifth step follows by the assumption on the function $\text{SAMPLE}(\cdot)$ in Equation (5.1).
- The last step follows by the definition of function $\text{SIMSTEP}(\cdot)$. □

```

1: function ESTIMATE( $\langle P, \mathbf{d} \rangle_{\mathcal{E}}, k, N$ )
2:    $\forall i : (0 \leq i \leq k) : E_i \leftarrow \emptyset$ 
3:    $counter \leftarrow 0$ 
4:   while  $counter < N$  do
5:      $(c_0, \dots, c_k) \leftarrow \text{SIMULATE}(\langle P, \mathbf{d} \rangle_{\mathcal{E}}, k)$ 
6:      $\forall i : E_i \leftarrow E_i, c_i$ 
7:      $counter \leftarrow counter + 1$ 
8:   end while
9:   return  $E_0, \dots, E_k$ 
10: end function

```

Figure 3. Function used to obtain N samples of the evolution sequence of a configuration.

To compute the empirical evolution sequence of a configuration c the function ESTIMATE in Figure 3 can be used.

The function ESTIMATE(c, k, N) invokes N times the function SIMULATE in Figure 2 to sample a sequence of configurations c_0, \dots, c_k , modelling k steps of a computation from $c = c_0$. Then, a sequence of observations E_0, \dots, E_k is computed, where each E_i is the tuple c_i^1, \dots, c_i^N of configurations observed at time i in each of the N sampled computations.

Notice that, for each $i \in \{0, \dots, k\}$, the samples c_i^1, \dots, c_i^N are independent and moreover, by Lemma 5.2, they are identically distributed. Each E_i can be used to estimate the distribution $\mathcal{S}_{c,i}^C$. For any i , with $0 \leq i \leq k$, we let $\hat{\mathcal{S}}_{c,i}^{C,N}$ be the distribution such that for any measurable set of configurations $\mathbb{C} \in \Sigma_C$ we have

$$\hat{\mathcal{S}}_{c,i}^{C,N}(\mathbb{C}) = \frac{|E_i \cap \mathbb{C}|}{N}.$$

Finally, we let $\hat{\mathcal{S}}_c^{D,N} = \hat{\mathcal{S}}_{c,0}^{D,N} \dots \hat{\mathcal{S}}_{c,k}^{D,N}$ be the *empirical evolution sequence* such that for any measurable set of data states $\mathbb{D} \in \mathcal{B}_D$ we have

$$\hat{\mathcal{S}}_{c,i}^{D,N}(\mathbb{D}) = \hat{\mathcal{S}}_{c,i}^{C,N}(\langle \mathcal{P}, \mathbb{D} \rangle_{\mathcal{E}}).$$

Then, by applying the weak law of large numbers to the i.i.d samples, we get that when N goes to infinite both $\hat{\mathcal{S}}_{c,i}^{C,N}$ and $\hat{\mathcal{S}}_{c,i}^{D,N}$ converge weakly to $\mathcal{S}_{c,i}^C$ and $\mathcal{S}_{c,i}^D$ respectively:

$$\lim_{N \rightarrow \infty} \hat{\mathcal{S}}_{c,i}^{C,N} = \mathcal{S}_{c,i}^C \quad \lim_{N \rightarrow \infty} \hat{\mathcal{S}}_{c,i}^{D,N} = \mathcal{S}_{c,i}^D. \quad (5.2)$$

The algorithms in Figures 2 and 3 have been implemented in Python to support analysis of systems. The tool is available at <https://github.com/quasylab/spear>.

Example 5.3. A simulation of the three-tanks laboratory experiment is given in Figure 4, with the following setting: (i) $l_{min} = 0$, (ii) $l_{max} = 20$, (iii) $l_{goal} = 10$, (iv) $\Delta_t = 0.5$, (v) $q_{max} = 6$, (vi) $q_{step} = q_{max}/5$, (vii) $\Delta_q = 0.5$, (viii) $\Delta\tau = 0.1$. Moreover, we assume an initial data state \mathbf{d}_0 with $l_i = l_{min}$ and $q_i = 0$, for $i = 1, 2, 3$. The plot on the left hand side of Figure 4 depicts a simulation run of the system \mathbf{s}_1 , namely the system in which the environment evolution follows scenario 1 and having $c_1 = \langle \mathbf{P}_{\text{Tanks}}, \mathbf{d}_0 \rangle_{\mathcal{E}_1}$ as initial configuration. On the right hand side we have the corresponding plot for system \mathbf{s}_2 , whose environment evolution follows scenario 2 and the initial configuration is thus $c_2 = \langle \mathbf{P}_{\text{Tanks}}, \mathbf{d}_0 \rangle_{\mathcal{E}_2}$.

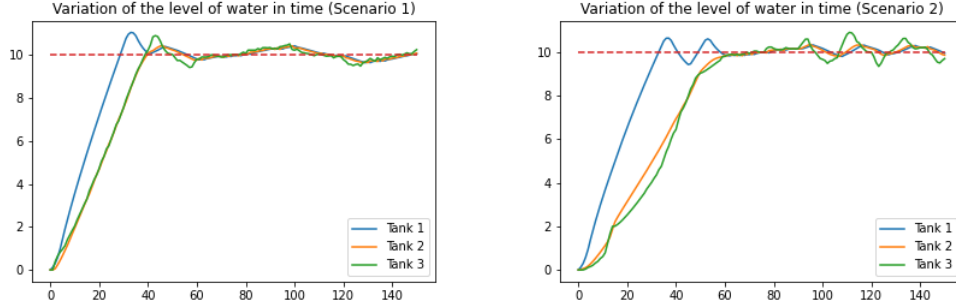


Figure 4. Simulation results. The dashed red line corresponds to $l_{goal} = 10$.

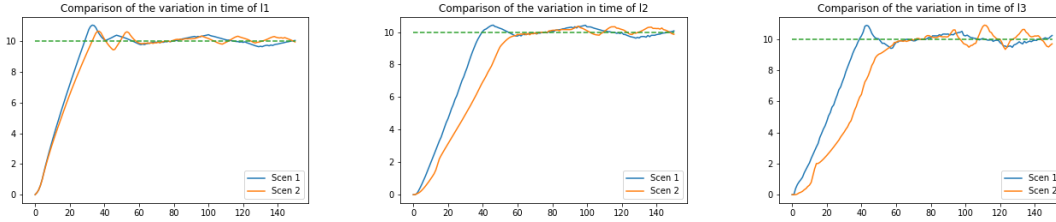


Figure 5. Simulation of the variation of the level of water in each tank in the two scenarios. The dashed green line corresponds to $l_{goal} = 10$.

In Figure 5 we consider the same simulations, but we compare the variations of the level of water, in time, in the same tank in the two scenarios. This can help us to highlight the potential differences in the evolution of the system with respect to the two environments.

In Figure 6 we give an estimation of the distributions of l_1, l_2, l_3 obtained from our simulations of \mathbf{s}_1 and \mathbf{s}_2 . In detail, the three plots in the top row report the comparison of the distributions over l_1 obtained in the two scenarios at time step, respectively, 50, 100 and 150 when using $N = 1000$ samples. The three plots in the central row are the corresponding ones for l_2 , and in the bottom row we have the plots related to l_3 . In both scenarios, we obtain Gaussian-like distributions, as expected given the probabilistic behaviour of the environment defined in Examples 3.2 and 3.10. However, we observe that while in both cases the mean of the distributions is close to $l_{goal} = 10$, there is a significant difference in their variance: the variance of the distributions obtained in scenario 1 is generally smaller than that of those in scenario 2 (the curves for scenario 1 are “slimmer” than those for scenario 2). We will see how this disparity is captured by the evolution metric. ◀

As usual in the related literature, we can apply the classic *standard error approach* to analyse the approximation error of our statistical estimation of the evolution sequences. Briefly, we let $x \in \{l_1, l_2, l_3\}$ and we focus on the distribution of the means of our samples (in particular we consider the cases $N = 1000, 5000, 10000$) for each variable x . In each case, for each $i \in \{0, \dots, k\}$ we compute the mean $\bar{E}_i(x) = \frac{1}{N} \sum_{j=1}^N \mathbf{d}_i^j(x)$ of the sampled data (recall that E_i is the tuple of the sampled configurations c_i^1, \dots, c_i^N , with $c_i^j = \langle P_i^j, \mathbf{d}_i^j \rangle \varepsilon$), and we evaluate their *standard deviation* $\tilde{\sigma}_{i,N}(x) = \sqrt{\frac{\sum_{j=1}^N (\mathbf{d}_i^j(x) - \bar{E}_i(x))^2}{N-1}}$ (see Figure 7a for

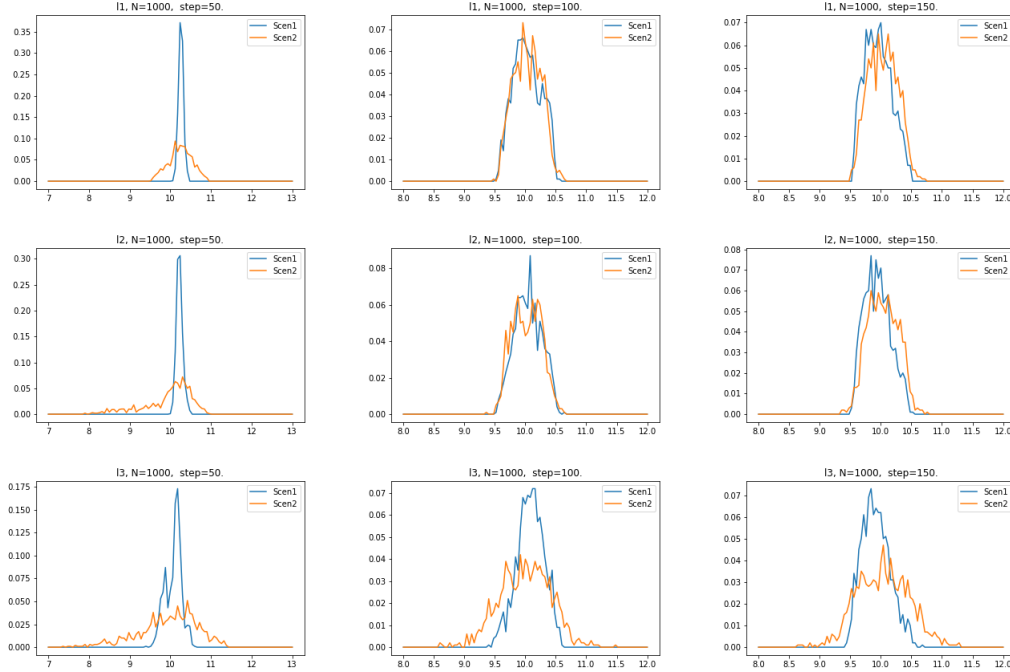


Figure 6. Estimated distributions of the water levels in the tanks in the two scenarios.

the variation in time of the standard deviation of the distribution of $x = l_3$). From $\tilde{\sigma}_{i,N}(x)$ we obtain the *standard error of the mean* $\bar{\sigma}_{i,N}(x) = \frac{\tilde{\sigma}_{i,N}(x)}{\sqrt{N}}$ (see Figure 7b for the variation in time of the standard error for the distribution of $x = l_3$). Finally, we proceed to compute the *z-score* of our sampled distribution as follows: $z_{i,N}(x) = \frac{\bar{E}_i(x) - \mathbb{E}(x)}{\bar{\sigma}_{i,N}(x)}$, where $\mathbb{E}(x)$ is the mean (or expected value) of the real distribution over x . In Figure 7c we report the variation in time of the *z-score* of the distribution over $x = l_3$: the dashed red lines correspond to $z = \pm 1.96$, namely the value of the *z-score* corresponding to a confidence interval of the 95%. We can see that our results can already be given with a 95% confidence in the case of $N = 1000$. Please notice that the oscillation in time of the values of the *z-scores* is due to the perturbations introduced by the environment in the simulations and by the natural oscillation in the interval $[l_{goal} - \Delta_l, l_{goal} + \Delta_l]$ of the water levels in the considered experiment (see Figure 4). A similar analysis, with analogous results, can be carried out for the distributions of l_1 and l_2 . In Figure 7d we report the variation in time of the *z-scores* of the distributions of the three variables, in the case $N = 1000$.

5.2. Computing distance between two configurations. Function ESTIMATE allows us to collect independent samples at each time step i from 0 to a deadline k . These samples can be used to estimate the distance between two configurations c_1 and c_2 . Following an approach similar to the one presented in [TK10], to estimate the Wasserstein distance $\mathbf{W}(m_{\rho,i}^{\mathcal{D}})$ between two (unknown) distributions $\mathcal{S}_{c_1,i}^{\mathcal{D}}$ and $\mathcal{S}_{c_2,i}^{\mathcal{D}}$ we can use N independent samples $\{c_1^1, \dots, c_1^N\}$ taken from $\mathcal{S}_{c_1,i}^{\mathcal{C}}$ and $\ell \cdot N$ independent samples $\{c_2^1, \dots, c_2^{\ell \cdot N}\}$ taken

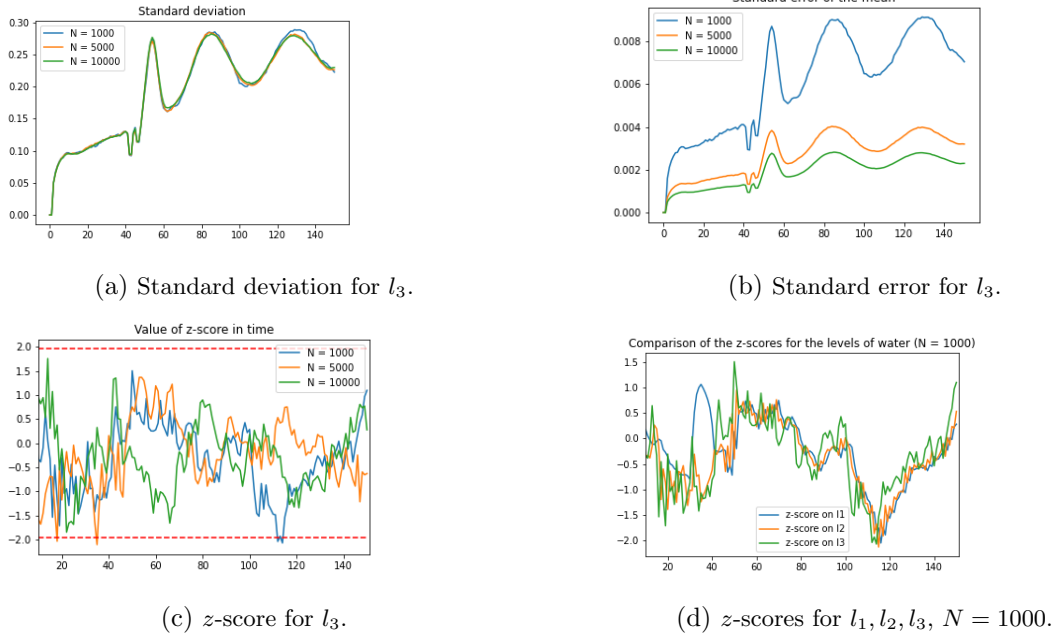


Figure 7. Analysis of the approximation error, over time, of the distributions of l_1, l_2, l_3 , in scenario 2, for $N = 1000, 5000, 10000$.

from $\mathcal{S}_{c_2,i}^C$. After that, we exploit the i -penalty function ρ_i and we consider the two sequences of values $\omega_1, \dots, \omega_N$ and $\nu_1, \dots, \nu_{\ell \cdot N}$ defined as follows:

$$\{\omega_j = \rho_i(\mathbf{d}_1^j) \mid \langle P_1^j, \mathbf{d}_1^j \rangle_{\mathcal{E}_1} = c_1^j\} \quad \{\nu_h = \rho_i(\mathbf{d}_2^h) \mid \langle P_2^h, \mathbf{d}_2^h \rangle_{\mathcal{E}_2} = c_2^h\}.$$

We can assume, without loss of generality that these sequences are ordered, i.e., $\omega_j \leq \omega_{j+1}$ and $\nu_h \leq \nu_{h+1}$. The value $\mathbf{W}(m_{\rho,i}^D)(\mathcal{S}_{c_1,i}^D, \mathcal{S}_{c_2,i}^D)$ can be approximated as $\frac{1}{\ell N} \sum_{h=1}^{\ell N} \max\{\nu_h - \omega_{\lceil \frac{h}{\ell} \rceil}, 0\}$. The next theorem, based on results in [TK10, Vil08, Val74], ensures that the larger the number of samplings the closer the gap between the estimated value and the exact one.

Theorem 5.4. *Let $\mathcal{S}_{c_1,i}^C, \mathcal{S}_{c_2,i}^C \in \Pi(\mathcal{C}, \Sigma_{\mathcal{C}})$ be unknown. Let $\{c_1^1, \dots, c_1^N\}$ be independent samples taken from $\mathcal{S}_{c_1,i}^C$, and $\{c_2^1, \dots, c_2^{\ell \cdot N}\}$ independent samples taken from $\mathcal{S}_{c_2,i}^C$. Let $\{\omega_j = \rho_i(\mathbf{d}_1^j)\}$ and $\{\nu_h = \rho_i(\mathbf{d}_2^h)\}$ be the ordered sequences obtained from the samples and the i -penalty function. Then, the Wasserstein distance between $\mathcal{S}_{c_1,i}^D$ and $\mathcal{S}_{c_2,i}^D$ is equal, a.s., to*

$$\mathbf{W}(m_{\rho,i}^D)(\mathcal{S}_{c_1,i}^D, \mathcal{S}_{c_2,i}^D) = \lim_{N \rightarrow \infty} \frac{1}{\ell N} \sum_{h=1}^{\ell N} \max\left\{\nu_h - \omega_{\lceil \frac{h}{\ell} \rceil}, 0\right\}.$$

Proof. Let $c_1^j = \langle P_1^j, \mathbf{d}_1^j \rangle_{\mathcal{E}}$, for all $j = 1, \dots, N$, and $c_2^j = \langle P_2^j, \mathbf{d}_2^j \rangle_{\mathcal{E}}$, for all $j = 1, \dots, \ell \cdot N$. We split the proof into two parts showing respectively:

$$\mathbf{W}(m_{\rho,i}^D)(\mathcal{S}_{c_1,i}^D, \mathcal{S}_{c_2,i}^D) = \lim_{N \rightarrow \infty} \mathbf{W}(m_{\rho,i}^D)(\hat{\mathcal{S}}_{c_1,i}^{\mathcal{D},N}, \hat{\mathcal{S}}_{c_2,i}^{\mathcal{D},\ell N}) . \quad (5.3)$$

$$\mathbf{W}(m_{\rho,i}^D)(\hat{\mathcal{S}}_{c_1,i}^{\mathcal{D},N}, \hat{\mathcal{S}}_{c_2,i}^{\mathcal{D},\ell N}) = \frac{1}{\ell N} \sum_{h=1}^{\ell N} \max\left\{\nu_h - \omega_{\lceil \frac{h}{\ell} \rceil}, 0\right\} . \quad (5.4)$$

- PROOF OF EQUATION (5.3).

We recall that the sequence $\{\hat{\mathcal{S}}_{c_l,i}^{\mathcal{D},N}\}$ converges weakly to $\mathcal{S}_{c_l,i}^{\mathcal{D}}$ for $l \in \{1,2\}$ (see Equation (5.2)). Moreover, we can prove that these sequences converge weakly in $\Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$ in the sense of [Vil08, Definition 6.8]. In fact, given the i -ranking function ρ_i , the existence of a data state $\tilde{\mathbf{d}}$ such that $\rho_i(\tilde{\mathbf{d}}) = 0$ is guaranteed (remember that the constraints used to define ρ_i are on the possible values of state variables and a data state fulfilling all the requirements is assigned value 0). Thus, for any $\mathbf{d} \in \mathcal{D}$ we have that

$$m_{\rho_i}^{\mathcal{D}}(\tilde{\mathbf{d}}, \mathbf{d}) = \max\{\rho_i(\mathbf{d}) - \rho_i(\tilde{\mathbf{d}}), 0\} = \rho_i(\mathbf{d}) .$$

Since, moreover, by definition ρ_i is continuous and bounded, the weak convergence of the distributions gives

$$\begin{aligned} \int_{\mathcal{D}} \rho_i(\mathbf{d}) d(\hat{\mathcal{S}}_{c_1,i}^{\mathcal{D},N}(\mathbf{d})) &\rightarrow \int_{\mathcal{D}} \rho_i(\mathbf{d}) d(\mathcal{S}_{c_1,i}^{\mathcal{D}}(\mathbf{d})) \\ \int_{\mathcal{D}} \rho_i(\mathbf{d}) d(\hat{\mathcal{S}}_{c_2,i}^{\mathcal{D},\ell N}(\mathbf{d})) &\rightarrow \int_{\mathcal{D}} \rho_i(\mathbf{d}) d(\mathcal{S}_{c_2,i}^{\mathcal{D}}(\mathbf{d})) \end{aligned}$$

and thus Definition 6.8.(i) of [Vil08] is satisfied. As \mathcal{D} is a Polish space, by [Vil08, Theorem 6.9] we obtain that

$$\hat{\mathcal{S}}_{c_l,i}^{\mathcal{D},N} \rightarrow \mathcal{S}_{c_l,i}^{\mathcal{D}} \text{ implies } \mathbf{W}(m_{\rho_i}^{\mathcal{D}})(\mathcal{S}_{c_1,i}^{\mathcal{D}}, \mathcal{S}_{c_2,i}^{\mathcal{D}}) = \lim_{N \rightarrow \infty} \mathbf{W}(m_{\rho_i}^{\mathcal{D}})(\hat{\mathcal{S}}_{c_1,i}^{\mathcal{D},N}, \hat{\mathcal{S}}_{c_2,i}^{\mathcal{D},\ell N}) .$$

- PROOF OF EQUATION (5.4).

For this part of the proof we follow [TK10]. Since the ranking function is continuous, it is in particular $\mathcal{B}_{\mathcal{D}}$ measurable and therefore for any distribution μ on $(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$ we obtain that

$$F_{\mu, \rho_i}(r) := \mu(\{\rho_i(\mathbf{d}) < r\})$$

is a well defined cumulative distribution function. In particular, for $\mu = \hat{\mathcal{S}}_{c_1,i}^{\mathcal{D},N}$ we have that

$$F_{\hat{\mathcal{S}}_{c_1,i}^{\mathcal{D},N}, \rho_i}(r) = \hat{\mathcal{S}}_{c_1,i}^{\mathcal{D},N}(\{\rho_i(\mathbf{d}) < r\}) = \frac{|\{\langle P_1^j, \mathbf{d}_1^j \rangle_{\mathcal{E}_1} \in E_{1,i} \mid \rho_i(\mathbf{d}_1^j) < r\}|}{N} .$$

Since, moreover, we can always assume that the values $\rho_i(\mathbf{d}_1^j)$ are sorted, so that $\rho_i(\mathbf{d}_1^j) \leq \rho_i(\mathbf{d}_1^{j+1})$ for each $j = 1, \dots, N-1$, we can express the counter image of the cumulative distribution function as

$$F_{\hat{\mathcal{S}}_{c_1,i}^{\mathcal{D},N}, \rho_i}^{-1}(r) = \rho_i(\mathbf{d}_1^j) \text{ whenever } \frac{j-1}{N} < r \leq \frac{j}{N} . \quad (5.5)$$

A similar reasoning holds for $F_{\hat{\mathcal{S}}_{c_2,i}^{\mathcal{D},\ell N}, \rho_i}^{-1}(r)$.

Then, by [FR18, Proposition 6.2], for each N we have that

$$\mathbf{W}(m_{\rho_i}^{\mathcal{D}})(\hat{\mathcal{S}}_{c_1,i}^{\mathcal{D},N}, \hat{\mathcal{S}}_{c_2,i}^{\mathcal{D},\ell N}) = \int_0^1 \max \left\{ F_{\hat{\mathcal{S}}_{c_2,i}^{\mathcal{D},\ell N}, \rho_i}^{-1}(r) - F_{\hat{\mathcal{S}}_{c_1,i}^{\mathcal{D},N}, \rho_i}^{-1}(r), 0 \right\} dr .$$

Let us now partition the interval $[0, 1]$ into ℓN intervals of size $\frac{1}{\ell N}$, thus obtaining

$$\mathbf{W}(m_{\rho_i}^{\mathcal{D}})(\hat{\mathcal{S}}_{c_1,i}^{\mathcal{D},N}, \hat{\mathcal{S}}_{c_2,i}^{\mathcal{D},\ell N}) = \sum_{h=1}^{\ell N} \left(\int_{\frac{h-1}{\ell N}}^{\frac{h}{\ell N}} \max \left\{ F_{\hat{\mathcal{S}}_{c_2,i}^{\mathcal{D},\ell N}, \rho_i}^{-1}(r) - F_{\hat{\mathcal{S}}_{c_1,i}^{\mathcal{D},N}, \rho_i}^{-1}(r), 0 \right\} dr \right) .$$

| | |
|---|---|
| <pre> 1: function DISTANCE($c_1, c_2, \rho, \lambda, \text{OT}, N, \ell$) 2: $k \leftarrow \max\{i \in \text{OT}\}$ 3: $E_{1,1}, \dots, E_{1,k} \leftarrow \text{ESTIMATE}(c_1, k, N)$ 4: $E_{2,1}, \dots, E_{2,k} \leftarrow \text{ESTIMATE}(c_2, k, \ell N)$ 5: $m \leftarrow 0$ 6: for all $i \in \text{OT}$ do 7: $m_i \leftarrow \text{COMPUTE W}(E_{1,i}, E_{2,i}, \rho_i)$ 8: $m \leftarrow \max\{m, \lambda(i) \cdot m_i\}$ 9: end for 10: return m 11: end function </pre> | <pre> 1: function COMPUTEW(E_1, E_2, ρ) 2: $(\langle P_1^1, \mathbf{d}_1^1 \rangle_{\mathcal{E}_1}, \dots, \langle P_1^N, \mathbf{d}_1^N \rangle_{\mathcal{E}_1}) \leftarrow E_1$ 3: $(\langle P_2^1, \mathbf{d}_2^1 \rangle_{\mathcal{E}_2}, \dots, \langle P_2^{\ell N}, \mathbf{d}_2^{\ell N} \rangle_{\mathcal{E}_2}) \leftarrow E_2$ 4: $\forall j : (1 \leq j \leq N) : \omega_j \leftarrow \rho(\mathbf{d}_1^j)$ 5: $\forall h : (1 \leq h \leq \ell N) : \nu_h \leftarrow \rho(\mathbf{d}_2^h)$ 6: re index $\{\omega_j\}$ s.t. $\omega_j \leq \omega_{j+1}$ 7: re index $\{\nu_h\}$ s.t. $\nu_h \leq \nu_{h+1}$ 8: return $\frac{1}{\ell N} \sum_{h=1}^{\ell N} \max\{\nu_h - \omega_{\lceil \frac{h}{\ell} \rceil}, 0\}$ 9: end function </pre> |
|---|---|

Figure 8. Functions used to estimate the evolution metric on systems.

From Equation (5.5), on each interval $(\frac{h-1}{\ell N}, \frac{h}{\ell N}]$ it holds that $F_{\hat{\mathcal{S}}_{c_1, i}^{\mathcal{D}, N}, \rho_i}^{-1}(r) = \rho_i(\mathbf{d}_1^{\lceil \frac{h}{\ell} \rceil})$ and $F_{\hat{\mathcal{S}}_{c_2, i}^{\mathcal{D}, \ell N}, \rho_i}^{-1}(r) = \rho_i(\mathbf{d}_2^h)$. Moreover, both functions are constant on each the interval so that the value of the integral is given by the difference multiplied by the length of the interval:

$$\begin{aligned} \mathbf{W}(m_{\rho, i}^{\mathcal{D}}(\hat{\mathcal{S}}_{c_1, i}^{\mathcal{D}, N}, \hat{\mathcal{S}}_{c_2, i}^{\mathcal{D}, \ell N})) &= \sum_{h=1}^{\ell N} \frac{1}{\ell N} \max \left\{ \rho_i(\mathbf{d}_2^h) - \rho_i(\mathbf{d}_1^{\lceil \frac{h}{\ell} \rceil}), 0 \right\} \\ &= \sum_{h=1}^{\ell N} \frac{1}{\ell N} \max \left\{ \nu_h - \omega_{\lceil \frac{h}{\ell} \rceil}, 0 \right\} . \end{aligned}$$

By substituting the last equality into Equation (5.3) we obtain the thesis. \square

The procedure outlined above is realised by functions DISTANCE and COMPUTEW in Figure 8. The former takes as input the two configurations to compare, the penalty function (seen as the sequence of the i -penalty functions), the discount function λ , the set OT of observation times which we assume to be bounded, and the parameters N and ℓ used to obtain the samplings of computation.

Function DISTANCE collects the samples E_i of possible computations during the observation period $[0, \max_{\text{OT}}]$, where \max_{OT} denotes the last observation time.

Then, for each observation time $i \in \text{OT}$, the distance at time i is computed via the function COMPUTEW($E_{1,i}, E_{2,i}, \rho_i$).

Since the penalty function allows us to reduce the evaluation of the Wasserstein distance in \mathbb{R}^n to its evaluation on \mathbb{R} , due to the sorting of $\{\nu_h \mid h \in [1, \dots, \ell N]\}$ the complexity of function COMPUTEW is $O(\ell N \log(\ell N))$ (cf. [TK10]). We refer the interested reader to [SFG⁺21, Corollary 3.5, Equation (3.10)] for an estimation of the approximation error given by the evaluation of the Wasserstein distance over N samples.

5.3. Analysis of the three-tanks experiment. Our aim is now to use the evolution metric, and the algorithms introduced above, to study various properties of the behaviour of the three-tanks system. In particular, we consider the following two objectives:

- (1) Comparison of the behaviour of two systems generated from the interaction of the same program with two different environments, under the same initial conditions.
- (2) Comparison of the behaviour of two systems generated from the interaction of two different programs with the same environment, under the same initial conditions.

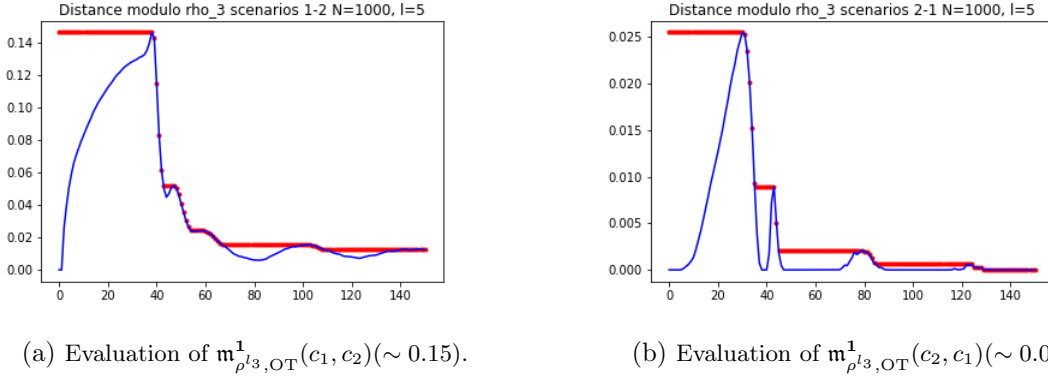


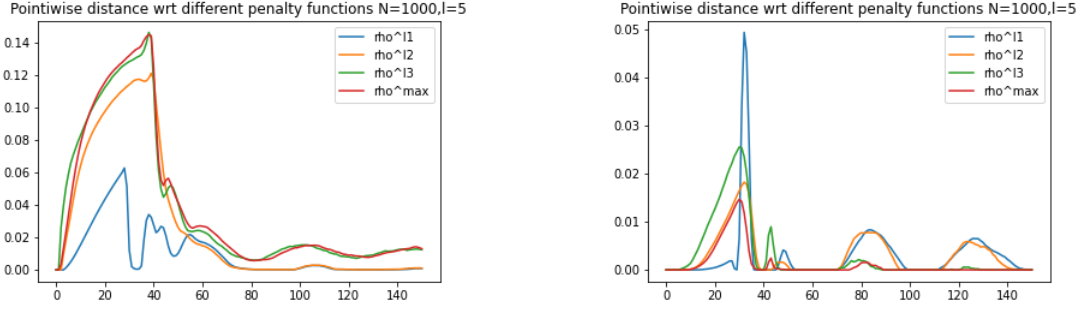
Figure 9. Evaluation of the distance between \mathbf{s}_1 and \mathbf{s}_2 (and $\mathbf{s}_2, \mathbf{s}_1$) with respect to the target $l_3 = l_{goal}$. A red dot at coordinates (x, y) means that $m_{\rho^{l_3}, OT_x}^1(c_1, c_2) = y$, where $OT_x = \{\tau \in OT \mid \tau \geq x\}$. The blue line is the pointwise Wasserstein distance between the distributions in the evolution sequences of c_i and c_{3-i} .

5.3.1. *Same program, different environments.* Consider the systems \mathbf{s}_1 and \mathbf{s}_2 introduced in Example 5.3. We recall that the initial configuration of \mathbf{s}_i is $c_i = \langle P_{\text{Tanks}}, \mathbf{d}_0 \rangle_{\mathcal{E}_i}$, for $i = 1, 2$. Notice that since \mathbf{s}_1 and \mathbf{s}_2 are distinguished only by the environment function, a comparison of their behaviour by means of the evolution metric allows us to establish in which scenario the program P_{Tanks} performs better with respect to the aforementioned tasks.

Firstly, we focus on a single tank by considering the target $l_3 = l_{goal}$, and thus the penalty function ρ^{l_3} . We do not consider any discount (so λ is the constant function $\mathbf{1}$, i.e., $\lambda(\tau) = 1$ for all τ) and we let $OT = \mathbb{N} \cap [0, 150]$. In Figure 9 we show the evaluation of $m_{\rho^{l_3}, OT}^1$ between \mathbf{s}_1 and \mathbf{s}_2 (Figure 9a) and between \mathbf{s}_2 and \mathbf{s}_1 (Figure 9b) over a simulation with 5000 samples ($N = 1000, l = 5$). Notice that the first distance is $m_{\rho^{l_3}, OT}^1(c_1, c_2) \sim 0.15$, whereas the second one is $m_{\rho^{l_3}, OT}^1(c_2, c_1) \sim 0.02$. The different scale of the two distances (the latter is less than $1/7$ of the former) already gives us an intuition of the advantages the use of a hemimetric gives us over a (pseudo)metric in terms of expressiveness.

Consider Figure 9a. Recall that, given the definition of our hemimetric on data states, the value of $m_{\rho^{l_3}, OT}^1(c_1, c_2)$ expresses how much the probabilistic evolution of the data in the evolution sequence of c_2 is worse than that of c_1 , with respect to the task identified by ρ^3 . In other words, the higher the value of $m_{\rho^{l_3}, OT}^1(c_1, c_2)$, the worse the performance of c_2 with respect to c_1 . Yet, for those who are not familiar with the Wasserstein distance, the plot alone can be a little foggy, so let us shed some light. Notice that, although after the first 40 steps the distance decreases considerably, it never reaches 0. This means that at each time step, the distribution in the evolution sequence of c_2 assigns positive probability to (at least) one measurable set of data states in which l_3 is farther from l_{goal} than in all the measurable sets of data states in the support of the distribution reached at the same time by c_1 . This is perfectly in line with our observations on the differences between the variances of the distributions in Figure 6 (cf. the bottom row pertaining l_3).

A natural question is then why the distance $m_{\rho^{l_3}, OT}^1(c_2, c_1)$, given in Figure 9b, is not equal to the constant 0. This is due to the combination of the Wasserstein distance with the hemimetric $m_{\rho^{l_3}}^{D_{3T}}$. Since $m_{\rho^{l_3}}^{D_{3T}}(\mathbf{d}_1, \mathbf{d}_2) = 0$ whenever $\rho^{l_3}(\mathbf{d}_2) < \rho^{l_3}(\mathbf{d}_1)$ and the Wasserstein



(a) Evaluation of $\mathbf{W}(m_\rho^{\mathcal{D}_{3T}})(\mathcal{S}_{c_1, \tau}^{\mathcal{D}_{3T}}, \mathcal{S}_{c_2, \tau}^{\mathcal{D}_{3T}})$ for $\rho \in \{\rho^{l_1}, \rho^{l_2}, \rho^{l_3}, \rho^{\max}\}$.

(b) Evaluation of $\mathbf{W}(m_\rho^{\mathcal{D}_{3T}})(\mathcal{S}_{c_2, \tau}^{\mathcal{D}_{3T}}, \mathcal{S}_{c_1, \tau}^{\mathcal{D}_{3T}})$ for $\rho \in \{\rho^{l_1}, \rho^{l_2}, \rho^{l_3}, \rho^{\max}\}$.

Figure 10. Evaluation of the evolution metric with respect to different targets.

lifting is defined as the infimum distance over the couplings, the (measurable sets of) data states reached by c_2 that are better, with respect to ρ^{l_3} , than those reached by c_1 , are “hidden” in the evaluation of $\mathbf{m}_{\rho^{l_3}, \text{OT}}^1(c_1, c_2)$. However, they do contribute to the evaluation of $\mathbf{m}_{\rho^{l_3}, \text{OT}}^1(c_2, c_1)$. Clearly, once we compute both $\mathbf{m}_{\rho^{l_3}, \text{OT}}^1(c_1, c_2)$ and $\mathbf{m}_{\rho^{l_3}, \text{OT}}^1(c_2, c_1)$, we can observe that the latter distance is less than 1/7 of the former and we can conclude thus that \mathbf{s}_1 shows a better behaviour than \mathbf{s}_2 .

As a final observation on Figure 9b, we notice that, for instance, in the time interval $I = [50, 70]$ the pointwise distance between the evolution sequences is 0. This means that for each (measurable set) data state \mathbf{d}_2 in the support of $\mathcal{S}_{c_2, \tau}^{\mathcal{D}_{3T}}$, $\tau \in I$, there is one (measurable set) data state \mathbf{d}_1 in the support of $\mathcal{S}_{c_1, \tau}^{\mathcal{D}_{3T}}$, such that $m_{\rho^{l_3}}^{\mathcal{D}_{3T}}(\mathbf{d}_2, \mathbf{d}_1) = 0$.

Clearly, one can repeat a similar analysis for the other tasks of the system. To give a general idea of the difference in the behaviour of the system in the two scenarios, in Figure 10 we report the evaluation of the pointwise distance between the evolution sequences of \mathbf{s}_1 and \mathbf{s}_2 (Figure 10a), and viceversa (Figure 10b).

5.3.2. Different programs, same environment. We introduce two new programs: $\mathbf{P}_{\text{Tanks}}^+$ and $\mathbf{P}_{\text{Tanks}}^-$. They are defined exactly as $\mathbf{P}_{\text{Tanks}}$, the only difference being the value of Δ_l used in the if/else guards in \mathbf{P}_{in} and \mathbf{P}_{out} . For $\mathbf{P}_{\text{Tanks}}$ we used $\Delta_l = 0.5$; we use $\Delta_l^+ = 0.7$ for $\mathbf{P}_{\text{Tanks}}^+$, and $\Delta_l^- = 0.3$ for $\mathbf{P}_{\text{Tanks}}^-$. Our aim is to compare the behaviour of \mathbf{s}_1 with that of systems \mathbf{s}^+ and \mathbf{s}^- having as initial configurations, respectively, $c^+ = \langle \mathbf{P}_{\text{Tanks}}^+, \mathbf{d}_0 \rangle_{\mathcal{E}_1}$ and $c^- = \langle \mathbf{P}_{\text{Tanks}}^-, \mathbf{d}_0 \rangle_{\mathcal{E}_1}$. Notice that the three systems are characterised by the same environment evolution \mathcal{E}_1 and the same initial data state \mathbf{d}_0 . Moreover, as outlined above, we see the value of Δ_l as an inherent property of the program.

So, while the outcome of the experiment will be exactly the one the reader expects, this example shows that with our framework we can compare the ability of different programs to fulfil their objectives when operating on equal footing, thus offering us the possibility of choosing the most suitable one according to (possibly) external factors. For instance, we can imagine that while Δ_l^- allows for a finer control, it also causes more frequent modifications to the flow rate q_1 than Δ_l does. These may in turn entail a higher risk of a breakdown of the pump attached to the first tank. In this case, we can use the evolution metric to

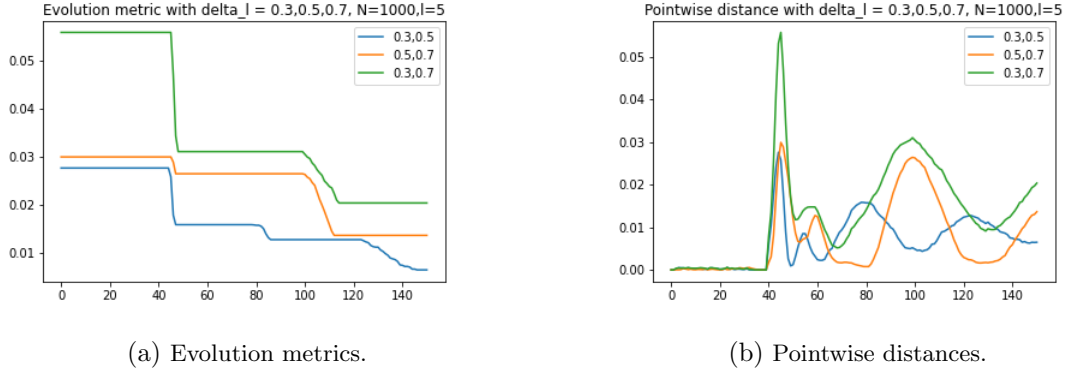


Figure 11. Evaluation of the distances between $P_{\text{Tanks}}^-, P_{\text{Tanks}}, P_{\text{Tanks}}^+$ with respect to ρ^{l_3} .

decide whether the difference in the performance of the programs justifies the higher risk, or if P_{Tanks} is still “good enough” for our purposes.

In Figure 11 we present the results of the evaluations of $\mathbf{m}_{\rho^{l_3}, \text{OT}}^1(c^-, c_1)$, $\mathbf{m}_{\rho^{l_3}, \text{OT}}^1(c_1, c^+)$, and $\mathbf{m}_{\rho^{l_3}, \text{OT}}^1(c^-, c^+)$, where the target of the programs is $l_3 = l_{\text{goal}}$ (i.e., the penalty function ρ^{l_3} is considered). In detail, in Figure 11a we report the values of the three evolution metrics (see Figure 9 for an explanation of how to read the graph), while in Figure 11b we report the pointwise (with respect to time) Wasserstein distances between the distributions in the evolution sequences of the systems.

6. ROBUSTNESS OF PROGRAMS

We devote this section to the analysis of the *robustness* of programs with respect to a data state and an environment. We also introduce two sensitivity properties of programs, which we call *adaptability* and *reliability*, entailing the ability of the program to induce a similar behaviour in systems that start operating from similar initial conditions.

6.1. Robustness. A system is said to be *robust* if it is able to function correctly even in the presence of uncertainties. In our setting, we formalise the *robustness* of programs by *measuring* their capability to *tolerate* perturbations in the environmental conditions.

First of all, we characterise the perturbations with respect to which we want the program to be robust. Given a program P , a data state \mathbf{d} , and an environment \mathcal{E} , we consider all data states \mathbf{d}' such that the behaviour of the original configuration $\langle P, \mathbf{d} \rangle_{\mathcal{E}}$ is at distance at most η_1 , for a chosen $\eta_1 \in [0, 1)$, from the behaviour of $\langle P, \mathbf{d}' \rangle_{\mathcal{E}}$. The distance is evaluated as the evolution metric with respect to a chosen penalty function ρ and a time interval I . The idea is that ρ and I can be used to characterise any particular situation we might want to study. For instance, if we think of a drone autonomously setting its trajectory, then I can be the time window within which a gust of wind occurs, and ρ can capture the distance from the intended trajectory.

We can then proceed to measure the robustness. Let ρ' be the penalty function related to the (principal) target of the system, and let $\tilde{\tau}$ be an observable time instant. We evaluate the evolution metric, with respect to ρ' , between the behaviour of $\langle P, \mathbf{d} \rangle_{\mathcal{E}}$ shown after time $\tilde{\tau}$,

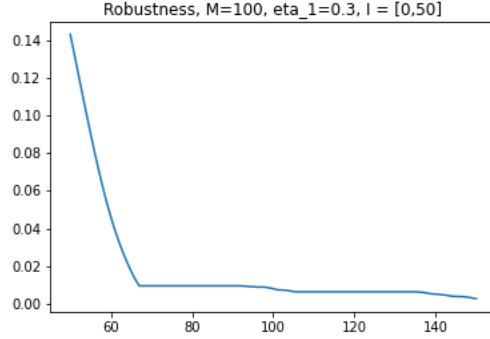


Figure 12. Robustness of P_{Tanks} with respect to \mathbf{d}_s and \mathcal{E}_1 , for $M = 100$, $\eta_1 = 0.3$, and $I = [0, 50]$.

and that of $\langle P, \mathbf{d}' \rangle_{\mathcal{E}}$ for all the perturbations \mathbf{d}' obtained above. We say that the robustness of P with respect to \mathbf{d} and \mathcal{E} is $\eta_2 \in [0, 1)$, if η_2 is an upper bound for all those distances.

Notice that we use two penalty functions, one to identify the perturbations (ρ), and one to measure the robustness (ρ'), which can be potentially related to different targets of the system. For instance, in the aforementioned case of the drone, ρ' can be related to the distance from the final location, or to battery consumption. This allows for an analysis of programs robustness in a general setting, where the perturbations and the (long term) system behaviour take into account different sets of variables. Moreover, the time instant $\tilde{\tau}$ plays the role of a *time bonus* given to the program to counter the effects of perturbations: differences detectable within time $\tilde{\tau}$ are not considered in the evaluation of the robustness threshold.

Definition 6.1 formalises the intuitions given above.

Definition 6.1 (Robustness). Let ρ, ρ' be two penalty functions, I a time interval, $\tilde{\tau} \in \text{OT}$, and $\eta_1, \eta_2 \in [0, 1)$. We say that P is $(\rho, \rho', I, \tilde{\tau}, \eta_1, \eta_2)$ -robust with respect to the data state \mathbf{d} and the environment evolution \mathcal{E} if

$$\forall \mathbf{d}' \in \mathcal{D} \text{ with } \mathbf{m}_{\rho, \text{OT} \cap I}^{\lambda}(\langle P, \mathbf{d} \rangle_{\mathcal{E}}, \langle P, \mathbf{d}' \rangle_{\mathcal{E}}) \leq \eta_1$$

it holds that

$$\mathbf{m}_{\rho', \{\tau \in \text{OT} \mid \tau \geq \tilde{\tau}\}}^{\lambda}(\langle P, \mathbf{d} \rangle_{\mathcal{E}}, \langle P, \mathbf{d}' \rangle_{\mathcal{E}}) \leq \eta_2 .$$

We can use our algorithm to measure the robustness of a given program. Given a configuration $\langle P, \mathbf{d} \rangle_{\mathcal{E}}$, a set OT of observation times, a given threshold $\eta_1 \geq 0$, a penalty function ρ , and a time interval I , we can sample M variations $\{\mathbf{d}_1, \dots, \mathbf{d}_M\}$ of \mathbf{d} , such that for any $i \in \{1, \dots, M\}$, $\mathbf{m}_{\rho, \text{OT} \cap I}^{\lambda}(\mathbf{d}, \mathbf{d}_i) \leq \eta_1$. Then, for each sampled data state \mathbf{d}_i we can estimate the distance between $c = \langle P, \mathbf{d} \rangle_{\mathcal{E}}$ and $c_i = \langle P, \mathbf{d}_i \rangle_{\mathcal{E}}$, with respect to ρ' , after time $\tilde{\tau}$, namely $\mathbf{m}_{\rho', \{\tau \in \text{OT} \mid \tau \geq \tilde{\tau}\}}^{\lambda}(c, c_i)$ for any $\tilde{\tau} \in \text{OT}$. Finally, for each $\tilde{\tau} \in \text{OT}$, we let

$$\xi_{\tilde{\tau}} = \max_{i \in \{1, \dots, M\}} \{ \mathbf{m}_{\rho', \{\tau \in \text{OT} \mid \tau \geq \tilde{\tau}\}}^{\lambda}(c, c_i) \} .$$

For the chosen η_1, ρ, I, ρ' and $\tilde{\tau}$, each $\xi_{\tilde{\tau}}$ gives us a lower bound to η_2 , and thus the robustness of the program.

Example 6.2. We can use our definition of robustness to show that whenever the program is able to (initially) keep the difference with respect to the target $l_3 = l_{goal}$ bounded, then also the distance with respect to the target $l_1 = l_{goal} = l_2$ will be bounded. Formally, we instantiate the parameters of Definition 6.1 as follows: $\rho = \rho^{l_3}$, $\rho' = \frac{1}{2}\rho^{l_1} + \frac{1}{2}\rho^{l_2}$, $I = [0, 50]$, and $\eta_1 = 0.3$. Figure 12 reports the evaluation of ξ_{50} with respect to $M = 100$ variations \mathbf{d}' satisfying $\mathbf{m}_{\rho, OT \cap I}^1(\langle P_{\text{Tanks}}, \mathbf{d}_s \rangle_{\mathcal{E}_1}, \langle P_{\text{Tanks}}, \mathbf{d}' \rangle_{\mathcal{E}_1}) \leq 0.3$. ◀

6.2. Adaptability and reliability. We define the sensitivity properties of *adaptability* and *reliability* as particular instances of the notion of robustness. Briefly, adaptability considers only one penalty function, and reduces the time interval I to the sole initial time step 0. Then, reliability also fixes $\tilde{\tau} = 0$. Let us present them in detail.

The notion of adaptability imposes some constraints on the *long term* behaviour of systems, disregarding their possible initial dissimilarities. Given the thresholds $\eta_1, \eta_2 \in [0, 1]$ and an observable time $\tilde{\tau}$, we say that a program P is adaptable with respect to a data state \mathbf{d} and an environment evolution \mathcal{E} if whenever P starts its computation from a data state \mathbf{d}' that differs from \mathbf{d} for at most η_1 , then we are guaranteed that the distance between the evolution sequences of the two systems after time $\tilde{\tau}$ is bounded by η_2 .

Definition 6.3 (Adaptability). Let ρ be a penalty function, $\tilde{\tau} \in OT$ and $\eta_1, \eta_2 \in [0, 1]$. We say that P is $(\tilde{\tau}, \eta_1, \eta_2)$ -adaptable with respect to the data state \mathbf{d} and the environment evolution \mathcal{E} if $\forall \mathbf{d}' \in \mathcal{D}$ with $m_{\rho, 0}^{\mathcal{D}}(\mathbf{d}, \mathbf{d}') \leq \eta_1$ it holds $\mathbf{m}_{\rho, \{\tau \in OT \mid \tau \geq \tilde{\tau}\}}^{\lambda}(\langle P, \mathbf{d} \rangle_{\mathcal{E}}, \langle P, \mathbf{d}' \rangle_{\mathcal{E}}) \leq \eta_2$.

We remark that one can always consider the data state \mathbf{d} as the ideal model of the world used for the specification of P , and the data state \mathbf{d}' as the real world in which P has to execute. Hence, the idea behind adaptability is that even if the initial behaviour of the two systems is quite different, P is able to reduce the gap between the real evolution and the desired one within the time threshold $\tilde{\tau}$. Notice that being $(\tilde{\tau}, \eta_1, \eta_2)$ -adaptable for $\tilde{\tau} = \min\{\tau \mid \tau \in OT\}$ is equivalent to being (τ, η_1, η_2) -adaptable for all $\tau \in OT$.

The notion of reliability strengthens that of adaptability by bounding the distance on the evolution sequences from the beginning. A program is reliable if it guarantees that small variations in the initial conditions cause only bounded variations in its evolution.

Definition 6.4 (Reliability). Let ρ be a penalty function and $\eta_1, \eta_2 \in [0, 1]$. We say that P is (η_1, η_2) -reliable with respect to the data state \mathbf{d} and the environment evolution \mathcal{E} if $\forall \mathbf{d}' \in \mathcal{D}$ with $m_{\rho, 0}^{\mathcal{D}}(\mathbf{d}, \mathbf{d}') \leq \eta_1$ it holds $\mathbf{m}_{\rho, OT}^{\lambda}(\langle P, \mathbf{d} \rangle_{\mathcal{E}}, \langle P, \mathbf{d}' \rangle_{\mathcal{E}}) \leq \eta_2$.

The same strategy depicted above for the evaluation of the robustness can be applied also in the case of adaptability and reliability. Briefly, we simulate the behaviour of M variations satisfying the requirement on the initial distance between data states $m_{\rho, 0}^{\mathcal{D}}(\mathbf{d}, \mathbf{d}_i) \leq \eta_1$, and use the bounds $\xi_{\tilde{\tau}}$ to estimate the adaptability bound η_2 . Similarly, for $\tau_{\min} = \min_{OT} \tau$, $\xi_{\tau_{\min}}$ gives a lower bound for its reliability.

Example 6.5. We provide an example of the evaluation of the adaptability of P_{Tanks} . We consider the environment evolution \mathcal{E}_1 , i.e., the one corresponding to scenario 1, and, as initial data state, the data state \mathbf{d}_s such that $\mathbf{d}_s(q_i) = 0$ and $\mathbf{d}_s(l_i) = 5$, for $i = 1, 2, 3$. The rationale to consider \mathbf{d}_s in place of \mathbf{d}_0 is that $\rho(\mathbf{d}_0) = 1$ and thus $m_{\rho}^{\mathcal{D}_{3T}}(\mathbf{d}_0, \mathbf{d}') = 0$ for all data states \mathbf{d}' , for any penalty function $\rho \in \{\rho^{l_1}, \rho^{l_2}, \rho^{l_3}, \rho^{\max}\}$. Conversely, there are several data states \mathbf{d}' such that $m_{\rho}^{\mathcal{D}_{3T}}(\mathbf{d}_s, \mathbf{d}') > 0$, so that we can take into account the behaviour

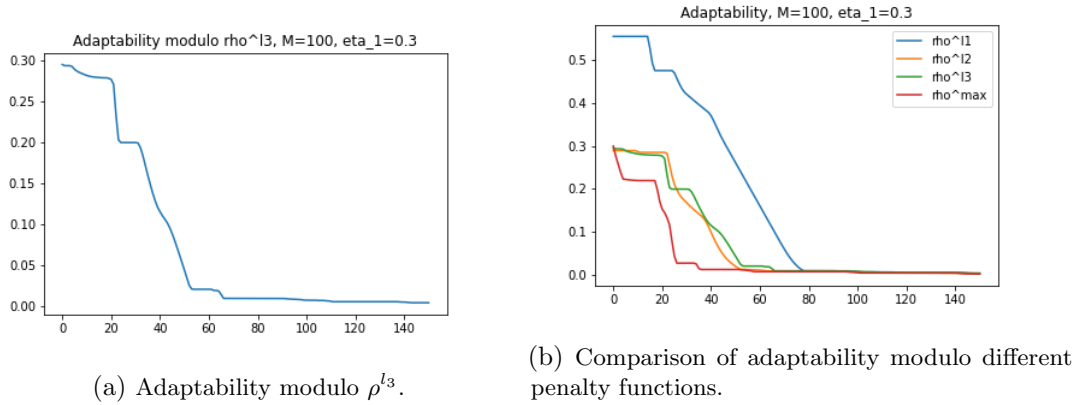


Figure 13. Adaptability of P_{Tanks} with respect to \mathbf{d}_s and \mathcal{E}_1 , for $M = 100$ and $\eta_1 = 0.3$.

of the system when starting from “worse” initial conditions. Let us stress that since the penalty functions ρ are evaluated only on (some of) the values l_i , for $i = 1, 2, 3$, the data states \mathbf{d}' that are within distance η_1 from \mathbf{d}_s can have any value in \mathcal{D}_q initially assigned to q_j , $j = 1, 2, 3$ (and any value in \mathcal{D}_l initially assigned to the l_j that are possibly not taken into account in the evaluation of ρ).

To obtain the results reported in Figure 13 we considered, for each $\rho \in \{\rho^{l_1}, \rho^{l_2}, \rho^{l_3}, \rho^{\max}\}$, $M = 100$ variations of the data state \mathbf{d}_s within the (closed) ball of radius $\eta_1 = 0.3$ with respect to $m_\rho^{\mathcal{D}^{3T}}$, and performed each simulation over 5000 samples ($N = 1000$, $\ell = 5$), keeping $k = 150$ as time horizon. Figure 13a represents the adaptability of P_{Tanks} with respect to \mathbf{d} and \mathcal{E}_1 for $\rho = \rho^{l_3}$. We can then infer that P_{Tanks} is $(30, 0.3, 0.2)$ -adaptable and $(50, 0.3, 0.05)$ -adaptable with respect to \mathbf{d}_s and \mathcal{E}_1 , when considering the target $l_3 = l_{\text{goal}}$. Please notice that one should always take into consideration the approximation error related to the computation of the Wasserstein distance [SFG⁺21, Corollary 3.5, Equation (3.10)] for an exact evaluation. Yet, in our presentation we are more interested in showing the method rather than making exact calculations. Figure 13b offers a comparison of the adaptability modulo the various penalty functions defined for the three-tanks system. ◀

7. CASE-STUDY: ENGINE SYSTEM

In this section we analyse a case study proposed in [LMMT21] and consisting in two supervised self-coordinating refrigerated engine systems that are subject to cyber-physical attacks aiming to inflict *overstress of equipment* [GGI⁺15].

Since the impact of the attacks can be viewed as a flaw on system behaviour, in our context it will be quantified by employing suitable penalty functions. In particular, the impact of attacks aiming to overstress the equipment can be quantified by adopting two different approaches: by simply measuring the level of equipment’s stress or by comparing such a value with the alarm signals generated by the system when attempting to perform attack detection. In the former case one focuses only on damages inflicted by attacks, in the latter case one is mainly interested in false positives and false negatives raised by the detection system. In the former case we will use penalty functions quantifying the level of

stress of system's equipment, in the latter case penalty functions will quantify false negatives and false positives that are produced when attempting to detect attacks and to raise alarm events. In both cases, our simulations will allow us to *estimate the distance between the genuine system and the system under attack*. Essentially, these simulations will allow us to estimate the impact of attacks, thus quantifying how worse the system under attack behaves with respect to the genuine one.

7.1. Modelling of the engine system. We start by describing a single engine. Then, we will describe the system obtained by combining two identical engines with a supervisor. Finally, we will introduce cyber-physical attacks tampering with a single engine or with the whole system. Interestingly, our choice of modelling the program and the environment separately well supports the specifications of these attacks: since the attacks consist in pieces of malicious code that are somehow injected in the logic of the system, an attack specified by a process A can be easily injected in a genuine system $\langle P, \mathbf{d} \rangle_{\mathcal{E}}$ by composing P and A in parallel, thus giving $\langle P \mid A, \mathbf{d} \rangle_{\mathcal{E}}$.

The engine we consider is a CPS whose logic aims to perform three tasks: (i) regulate the speed, (ii) maintain the temperature within a specific range by means of a cooling system, and (iii) detect anomalies. The first two tasks are on charge of a controller, the last one is took over by an intrusion detection system, henceforth IDS.

The data space consists of the following variables:

- i. *temp*, representing a sensor detecting the temperature of the engine;
- ii. *cool*, representing an actuator to turn on/off the cooling system;
- iii. *speed*, representing an actuator to regulate the engine speed, with values in the set $\{\text{slow}, \text{half}, \text{full}\}$, where *half* is the normal setting;
- iv. *ch_speed*, representing a channel used by the IDS to command the controller to set the actuator *speed* at *slow*, when an anomaly is detected, or at *half*, otherwise;
- v. *ch_in* and *ch_out*, representing two channels used for communication between the two engines: when an IDS commands to the controller of its own engine to operate at *slow* speed, in order to compensate the consequent lack of performance it asks to the controller of the other engine to operate at *full* speed;
- vi. *stress*, denoting the level of stress of the engine, due to its operating conditions;
- vii. six variables p_k , for $1 \leq k \leq 6$, recording the temperatures in the last six time instants;
- viii. *temp_fake*, representing a noise introduced by malicious activity of attackers tempering with sensor *temp*: we assume that the IDS can access the genuine value *temp*, whereas the controller receives the value of *temp* through an insecure channel *ch_temp* that may be compromised and whose value is obtained by summing up the noise *temp_fake* and *temp*;
- ix. *ch_warning*, representing a channel used by the IDS to raise anomalies.

The environment evolution \mathcal{E} affects these variables as follows:

- (1) *temp* changes by a value determined probabilistically according to a uniform distribution $\mathcal{U}(I)$, where I is the interval $[-1.2, -0.8]$ if the cooling system is active (*cool* = on), whereas if the cooling system is inactive (*cool* = off) then I depends on the speed: if *speed* is *slow* then $I = [0.1, 0.3]$, if *speed* is *half* then $I = [0.3, 0.7]$, if *speed* is *full* then $I = [0.7, 1.2]$;
- (2) the variables p_k , for $k \in 1 \dots 6$, are updated as expected to record the last six temperatures detected by sensor *temp*;

- (3) *stress* remains unchanged if the temperature was below a constant **max**, set to 100 in our experiments, for at least 3 of the last 6 time instants; otherwise, the variable is increased (reaching at most the value 1) by a constant **stressincr** that we set at 0.02;
- (4) all variables representing channels or actuators are not modified by \mathcal{E} , since they are under the control of the program.

Summarising, the dynamics of p_k and *stress* can be modelled via the following set of stochastic difference equations, with sampling time interval $\Delta\tau = 1$:

$$\begin{aligned} p_k(\tau + 1) &= \begin{cases} temp(\tau) & \text{if } k = 1 \\ p_{k-1}(\tau) & \text{if } k = 2, \dots, 6 \end{cases} \\ stress(\tau + 1) &= \begin{cases} stress(\tau) + \mathbf{stressincr} & \text{if } |\{k \mid p_k(\tau) \geq \mathbf{max}\}| > 3 \\ stress(\tau) & \text{otherwise} \end{cases} \end{aligned} \quad (7.1)$$

and *temp* varies by a value that is uniformly distributed in an interval depending on the state of actuators *cool* and *speed*:

$$\begin{aligned} temp(\tau + 1) &= temp(\tau) + v \\ v &\sim \begin{cases} \mathcal{U}[-1.2, -0.8] & \text{if } cool(\tau) = \text{on} \\ \mathcal{U}[0.1, 0.3] & \text{if } cool(\tau) = \text{off and } speed(\tau) = \text{slow} \\ \mathcal{U}[0.3, 0.7] & \text{if } cool(\tau) = \text{off and } speed(\tau) = \text{half} \\ \mathcal{U}[0.7, 1.2] & \text{if } cool(\tau) = \text{off and } speed(\tau) = \text{full.} \end{cases} \end{aligned} \quad (7.2)$$

The whole engine, the controller and the IDS are modelled by the following processes Eng, Ctrl and IDS, respectively:

$$\begin{aligned} \text{Eng} &\stackrel{def}{=} \text{Ctrl} \mid \text{IDS} \\ \text{Ctrl} &\stackrel{def}{=} \text{if } [ch_temp < \mathbf{threshold}] \sqrt{\cdot}. \text{Check else } (\text{on} \rightarrow \text{cool}). \text{Cooling} \\ \text{Cooling} &\stackrel{def}{=} \sqrt{\cdot}. \sqrt{\cdot}. \sqrt{\cdot}. \sqrt{\cdot}. \text{Check} \\ \text{Check} &\stackrel{def}{=} \text{if } [ch_speed = \text{slow}] ((\text{slow} \rightarrow \text{speed}), (\text{off} \rightarrow \text{cool})). \text{Ctrl} \\ &\quad \text{else } ((ch_in \rightarrow \text{speed}), (\text{off} \rightarrow \text{cool})). \text{Ctrl} \\ \text{IDS} &\stackrel{def}{=} \text{if } [temp > \mathbf{max} \wedge \text{cool} = \text{off}] \\ &\quad ((\text{hot} \rightarrow ch_warning), (\text{low} \rightarrow ch_speed), (\text{full} \rightarrow ch_out)). \text{IDS} \\ &\quad \text{else } ((\text{ok} \rightarrow ch_warning), (\text{half} \rightarrow ch_speed), (\text{half} \rightarrow ch_out)). \text{IDS} \end{aligned}$$

where we write $((e_1 \rightarrow x_1), \dots, (e_n \rightarrow x_n))$ to denote the prefix $(e_1 \dots e_n \rightarrow x_1 \dots x_n)$.

At each scan cycle Ctrl checks if the value of temperature as carried by channel *ch_temp* is too high, namely above **threshold**, which is a constant $\leq \mathbf{max}$ and set to 99.8 in our experiments. If this is the case, then Ctrl activates the coolant for 5 consecutive time instants, otherwise the coolant remains off. In both cases, before re-starting its scan cycle, Ctrl waits for instructions/requests to change the speed: if it receives instructions through channel *ch_speed* from the IDS to slow down the engine then it commands so through actuator *speed*. Otherwise, if the IDS of the other engine requests through channel *ch_in* to work at full of half power, then Ctrl sets *speed* accordingly.

The process IDS checks whether the cooling system is active when the temperature is above **max**. If this safety condition is violated then: (i) it raises the presence of an anomaly, via channel *ch_warning*; (ii) it commands Ctrl to slow down the engine, via *ch_speed*; (iii) it

requests to the other engine to run at full power, via channel ch_out . Otherwise, IDS asks (both local and external) controllers to set their engines at half power.

An engine system can be built by combining two engines, the “left” and the “right” one, interacting with a supervisory component that checks their correct functioning. The composite system can be defined as

$$\text{Eng.Sys} = \text{Eng.L} \mid \text{Eng.R} \mid \text{SV}$$

where processes Eng.L and Eng.R are obtained from Eng by renaming the variables as follows: (i) both ch_in in Eng.L and ch_out in Eng.R are renamed as $ch_speed_R_to_L$; (ii) both ch_in in Eng.R and ch_out in Eng.L are renamed as $ch_speed_L_to_R$; (iii) all remaining variables x used in Eng are renamed as x_L in Eng.L and as x_R in Eng.R . Then, we introduce a new channel ch_alarm which is used by SV to forward to the external environment the information obtained from the IDSs through $ch_warning_L$ and $ch_warning_R$. In detail, ch_alarm carries a value in the set $\{\text{none}, \text{left}, \text{right}, \text{both}\}$ depending on the anomaly warnings received from the left and right IDS:

$$\begin{aligned} \text{SV} &\stackrel{\text{def}}{=} \text{if } [ch_warning_L = \text{hot} \wedge ch_warning_R = \text{hot}] (\text{both} \rightarrow ch_alarm).\text{SV} \\ &\quad \text{else if } [ch_warning_L = \text{hot}] (\text{left} \rightarrow ch_alarm).\text{SV} \\ &\quad \quad \text{else if } [ch_warning_R = \text{hot}] (\text{right} \rightarrow ch_alarm).\text{SV} \\ &\quad \quad \text{else } (\text{none} \rightarrow ch_alarm).\text{SV} \end{aligned}$$

Following [LMMT21], cyber-physical attacks can be modelled as processes that run in parallel with systems under attack. Intuitively, they represent malicious code that has been injected in the logic of the system. Three examples of attack to Eng.Sys are the following, where $X \in \{L, R\}$:

$$\begin{aligned} \text{Att_Act}_X &\stackrel{\text{def}}{=} \text{if } [temp_X < \text{max} - \text{AC}] (\text{off} \rightarrow cool_X).\text{Att_Act}_X \text{ else } \surd.\text{Att_Act}_X \\ \text{Att_Sen}_X &\stackrel{\text{def}}{=} (-\text{TF} \rightarrow temp_fake_X).\text{Att_Sen}_X \\ \text{Att_Saw}_X &\stackrel{\text{def}}{=} ((\text{left} \rightarrow AW_l), (\text{right} \rightarrow AW_r)).\text{Att_Saw}_{X'} \\ \text{Att_Saw}_{X'} &\stackrel{\text{def}}{=} \text{if } [cs \in [AW_l, AW_r]] (-\text{TF} \rightarrow temp_fake_X).\text{Att_Saw}_{X'} \\ &\quad \text{else } (0 \rightarrow temp_fake_X).\text{Att_Saw}_{X'} \end{aligned}$$

Process Att_Act_X models an integrity attack on actuator $cool_X$. The five-instants cooling cycle started by Ctrl_X is maliciously interrupted as soon as $temp_X$ goes below $\text{max} - \text{AC}$, with AC a constant with value in $[1.0, 3.0]$, in order to let the temperature of the engine rise quickly above max , accumulating stress in the system, and requiring continuous activation of the cooling system. We recall that the quantification of stress accumulated by engine Eng_X is recorded in variable $stress_X$, which is incremented by \mathcal{E} when $|\{k \mid 1 \leq k \leq 6 \wedge p_k \geq \text{max}\}| > 3$. In [LMMT21] this attack is classified as *stealth*, since the condition $temp_X > \text{max} \wedge cool_X = \text{off}$ monitored by the IDS never holds. Notice that AC is a parameter of the attack.

Then, Att_Sen_X is an integrity attack to sensor $temp_X$. The attack adds a negative offset $-\text{TF}$, with TF a positive constant, to the temperature value carried by ch_temp_X , namely ch_temp_X becomes $temp_X - \text{TF}$. This attack aims to prevent some activation of the cooling system by Ctrl_X . Since the IDS raises a warning on $ch_warning_X$ each time we have $\text{max} < temp_X \leq \text{max} + \text{TF}$ and $cool_X = \text{off}$, this attack is not stealth. In this case, TF is a parameter of the attack.

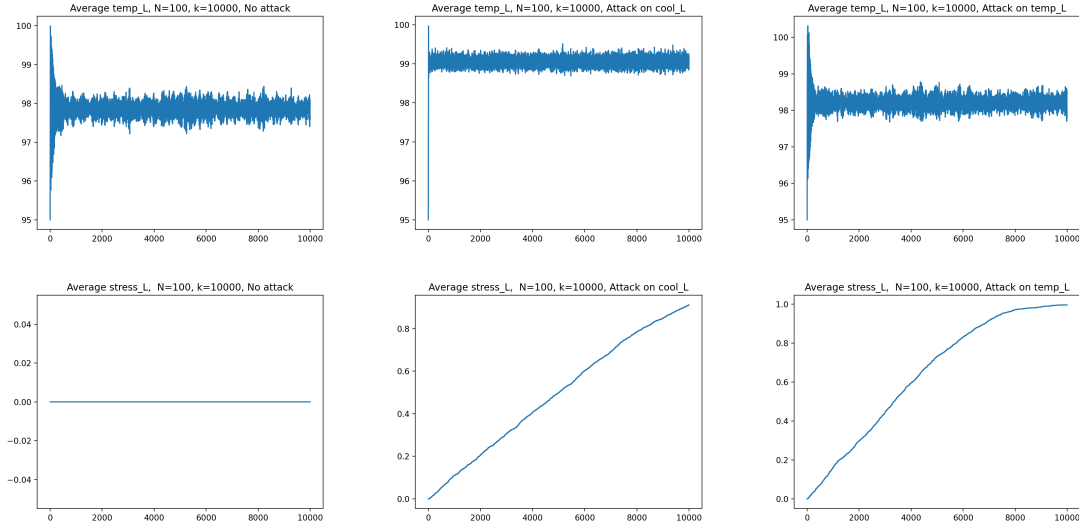


Figure 14. Average temperature (sensor $temp_L$) and stress (variable $stress_L$) for left engine running in three scenarios: no attack, attack on actuator $cool_L$, attack on sensor $temp_L$.

Simulation settings: $k = 10000$ steps, $N = 100$ runs.

Initial sensor/actuator settings: $temp_L = 95$, $cool_L = \text{off}$, $speed_L = \text{half}$.

Values assigned to constants: $AC = 1.8$, $TF = 0.4$.

Finally, Att_Saw_X performs the same attack of Att_Sen_X but only in a precise attack window. We assume that cs is a variable that simply counts the number of computation steps (the initial value is 0 and $cs(\tau + 1) = cs(\tau) + 1$ is added to Equation 7.1). Then, $[AW_l, AW_r]$ is an interval, where $AW_l = 0 = AW_r$ in the initial data state and we assume that the variables $left$ and $right$ take random non-negative integer values satisfying the property $0 \leq right - left \leq AWML$, where $AWML$ is a parameter of the attack representing the maximal length of the attack window.

7.2. Simulation of the engine system. We have applied our simulation strategy to the engine system. Figure 14 reports the results of six simulations of the single engine Eng_L , where each simulation is obtained by simulating 100 runs consisting in 10000 steps, with the following initial data state \mathbf{d}_0 : (i) $temp_L = p_1_L = \dots = p_6_L = 95$; (ii) $cool_L = \text{off}$; (iii) $speed_L = \text{half}$; (iv) $stress_L = 0$; (v) $ch_speed_L = ch_speed_L_to_R = ch_speed_R_to_L = \text{half}$; (vi) $temp_fake_L = 0$; and (vii) $ch_warning_L = \text{ok}$. We considered three different scenarios: (i) no attack, modelled by processes Eng_L , (ii) attack on actuator $cool_L$, modelled by $Eng_L \mid Att_Act_L$, with constant AC set to 1.8, and (iii) attack on sensor $temp_L$, modelled by $Eng_L \mid Att_Sen_L$, with constant TF set to 0.4. The six pictures report the value of average temperature detected by sensor $temp_L$ and the average stress level carried by variable $stress_L$ in these three scenarios.

Clearly, by comparing the plots in Figure 14 it turns out that both attacks are able to cause an average size-up of the temperature of the engine and to take the engine to the maximal level of stress after 10000 steps.

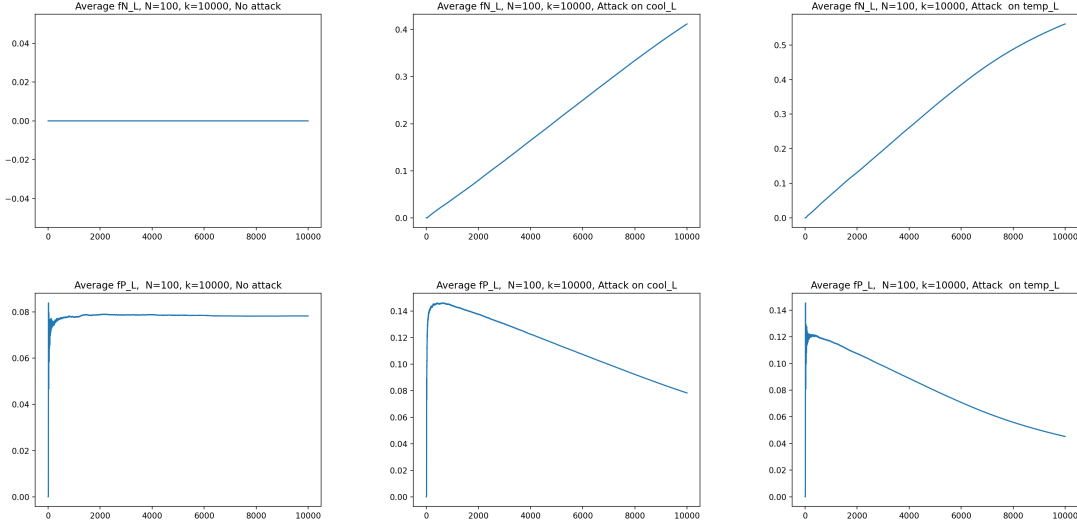


Figure 15. Average false negatives (variable fn_L) and false positives (variable fp_L) for the simulations of Figure 14.

7.3. Analysis of the engine system. Our aim is to use evolution metrics in order to study the impact of the attacks. This requires to compare the evolution sequences of the genuine system and those of the system under attack. The two approaches to the quantification of impact of attacks outlined above, consisting in evaluating only damages caused by attacks or both damages and alert signals, will be realised by employing different penalty functions.

We start with quantifying the impact of cyber-physical attacks by focusing on the runtime values of *false negatives* and *false positives*. Intuitively, if we focus on a single engine Eng_X , false negatives and false positives represent the average *effectiveness*, and the average *precision*, respectively, of the IDS modelled by IDS_X to signal through channel $ch_warning_X$ that the engine is under stress. In our setting, we can add two variables fn_X and fp_X that quantify false negatives and false positives depending on $stress_X$ and $ch_warning_X$. Both these variables are updated by \mathcal{E} . In detail, \mathcal{E} acts on fn_X and fp_X as follows:

$$fn_X(\tau + 1) = \frac{\tau * fn_X(\tau) + \max(0, stress_X(\tau) - ch_warning_X(\tau))}{\tau + 1}$$

$$fp_X(\tau + 1) = \frac{\tau * fp_X(\tau) + \max(0, ch_warning_X(\tau) - stress_X(\tau))}{\tau + 1}$$

where $ch_warning_X \in \{\text{hot}, \text{ok}\}$ is viewed as a real by setting $\text{hot} = 1$ and $\text{ok} = 0$. In this way, as reported in [LMMT21], fn_X and fp_X carry the average value, in probability and time, of the difference between the value of the stress of the system and of the warning raised by the IDS.

False negatives and false positives for the same simulations described in Figure 14 are reported in Figure 15.

Clearly, if we take the value of variable fn_X (resp. fp_X) as the value returned by our penalty function, we can quantify how worse Eng_X under attack behaves with respect to the genuine version of Eng_X , where the evaluation of these systems is done by taking into account their false negatives (resp. false positives). Notice that other approaches are

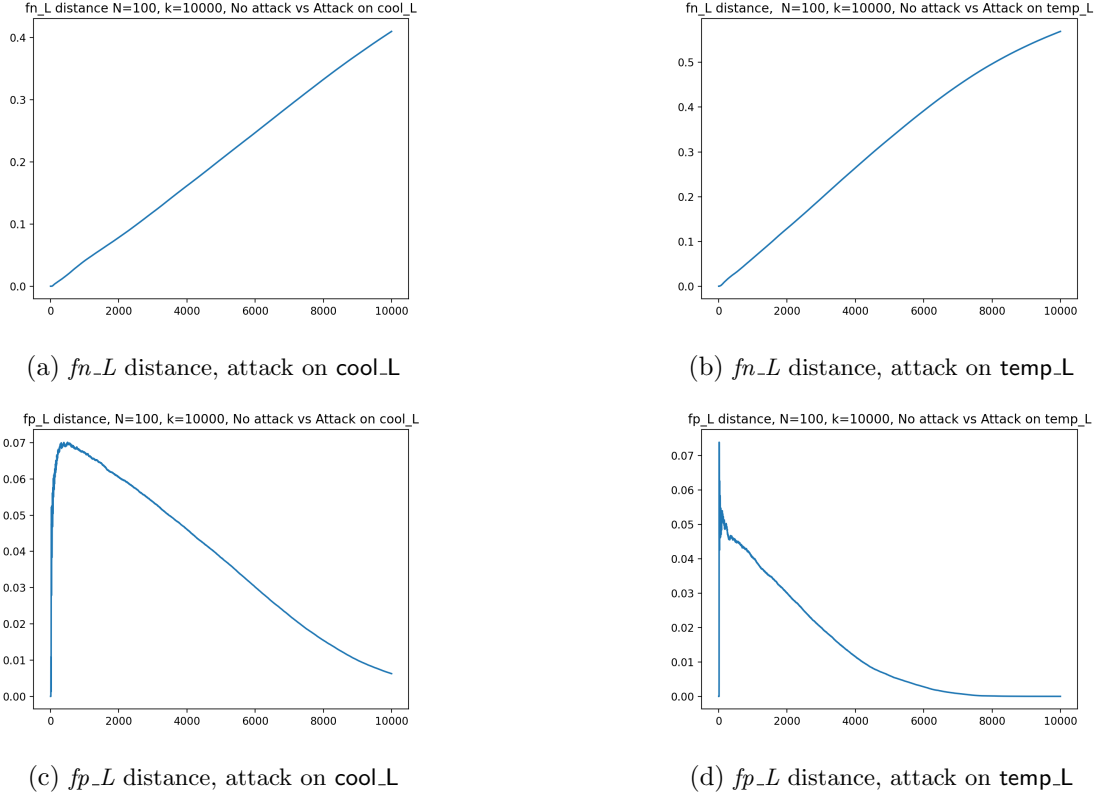


Figure 16. False negative and false positive distance between genuine Eng_L and Eng_L under actuator or sensor attack, for the simulations of Figure 14.

possible. For instance, by defining a penalty function returning a convex combination of the values of fn_X and fp_X , systems are evaluated by suitable weighting their false negatives and false positives.

In Figure 16 we report the distances between the genuine system Eng_L and the compromised systems Eng_L | Att_Act_L and Eng_L | Att_Sen_L in terms of false negatives and false positives. The settings for these simulations are the same of those used for the simulations described in Figure 14.

If we move to the whole system, the false negatives and false positives are carried by variables fn and fp , that are updated by \mathcal{E} as follows:

$$\begin{aligned} fn(\tau + 1) &= \frac{\tau * fn(\tau) + \min(1, (0.7 * \max(0, stress_L(\tau) - l) + 0.7 * \max(0, stress_R(\tau) - r)))}{\tau + 1} \\ fp(\tau + 1) &= \frac{\tau * fp(\tau) + \min(1, (0.7 * \max(0, l - stress_L(\tau)) + 0.7 * \max(0, r - stress_R(\tau))))}{\tau + 1} \end{aligned}$$

where $l = \begin{cases} 1 & \text{if } ch_alarm(\tau) \in \{\text{both, left}\} \\ 0 & \text{if } ch_alarm(\tau) \in \{\text{none, right}\} \end{cases}$ and $r = \begin{cases} 1 & \text{if } ch_alarm(\tau) \in \{\text{both, right}\} \\ 0 & \text{if } ch_alarm(\tau) \in \{\text{none, left}\} \end{cases}$.

Here, false negatives and false positives represent the average effectiveness, and the average precision, respectively, of the supervisors modelled by SV to signal through channel ch_alarm that the system is under stress. We remark that the values 0.7 are the values chosen in [LMMT21] to quantify the *security clearance* of the two engines, where, in general, the

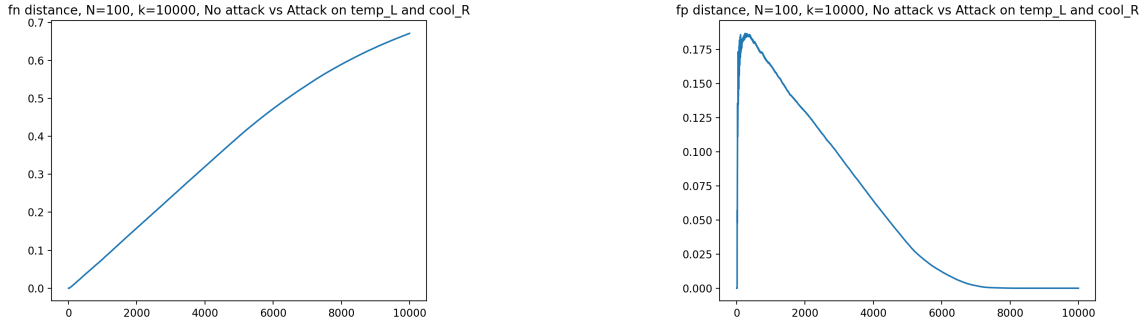


Figure 17. False negative and false positive distance between genuine `Eng_Sys` and `Eng_Sys` under attack on left sensor and right actuator.

clearance of a CPS component quantifies the importance of that component from a security point of view. In this case, the two engines have the same criticality and, consequently, the same security clearance.

In Figure 17 we report the distance, in terms of false negatives and false positives, between the system `Eng_Sys` and the compromised system `Eng_Sys | Att_Act.L | Att_Sen.R` that is subject to both an attack on the actuator `cool_L` of the left engine and an attack on the sensor `temp_R` of the right engine. Again, the settings for these simulations are the same of those used for simulations described in Figure 14.

Let us focus now on the third attack, namely the one implemented by process `Att_Saw_X`. Intuitively, we expect a correlation between the length of the attack window and the impact of the attack. In order to study such a relation, we may proceed as follows: (i) We define a penalty function ρ so that its value depends on the length of the attack window $[AW_l, AW_r]$. Actually, we may define ρ to return $(AW_r - AW_l)/AWML$. In particular, if we consider the genuine system then ρ returns 0, since $AW_l = 0 = AW_r$. (ii) We consider a penalty function ρ' expressing the impact of the attack. (iii) We estimate the $(\rho, \rho', [0, 0], 0, \eta_1, \eta_2)$ -robustness of the genuine system. More in detail, we may sample M variations of the system at ρ -distance from the genuine one bounded by η_1 . This means sampling M versions of `Eng_X | Att_Saw_X` characterised by attack windows of length bounded by $\eta_1 \cdot AWML$. Then, by applying our simulations, we can estimate how worse these compromised systems behave with respect to the genuine system `Eng_X`, according to the impact expressed by ρ' .

In Figure 18 we report the results of such an estimation for ρ' coinciding with fn_L , $[AW_l, AW_r] = [0, aw]$ an attack window of length aw starting at first instant, $AWML$ set to 1000 and $\eta_1 = 0.1$, which means that aw is chosen probabilistically according to a uniform distribution on interval $[0, 100]$.

Here we consider three different offset values assigned to TF: 0.4, 0.6, and 0.8. In all cases, we estimate to have robustness, for values for η_2 that are 0.012, 0.05, and 0.11, respectively.

Now, let us change the approach for attack quantification and assume that only the inflicted overstress is considered, without taking into account the raised alarm signals. In particular, this overstress is quantified at a given computation step n as the ratio between the number of computation steps where the stress condition $|\{k \mid p_k \geq \max\}| > 3$ holds and n . Therefore, the penalty function ρ' is defined now as the ratio between the values of two

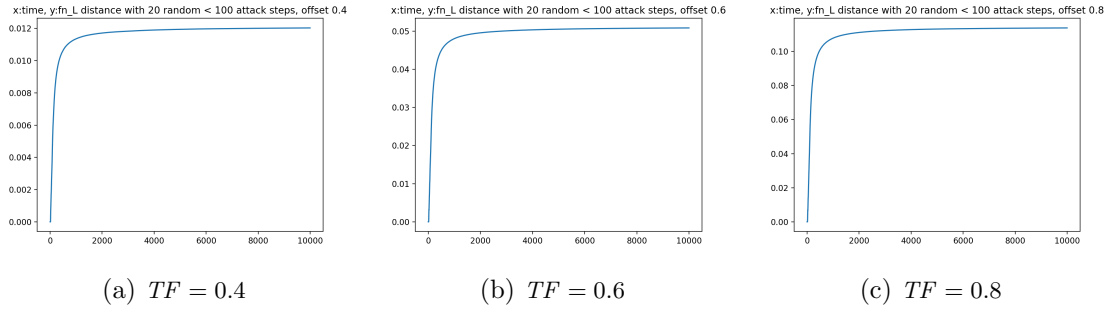


Figure 18. Estimation of false negative distance between a genuine engine and an engine subject to tampering with sensor $temp_L$. The attacker adds a negative offset -0.4 (Fig. 18a), or -0.6 (Fig. 18b), or -0.8 (Fig. 18c) to sensor, on the attack window $[0, aw]$, with aw an integer uniformly distributed in $[0, 100]$. Simulation settings for each offset: $M = 20$ different sampled values for each aw , $N = 100$ runs by the genuine engine and $N * l = 1000$ runs by the compromised system for each aw , $k = 10000$ steps per run.

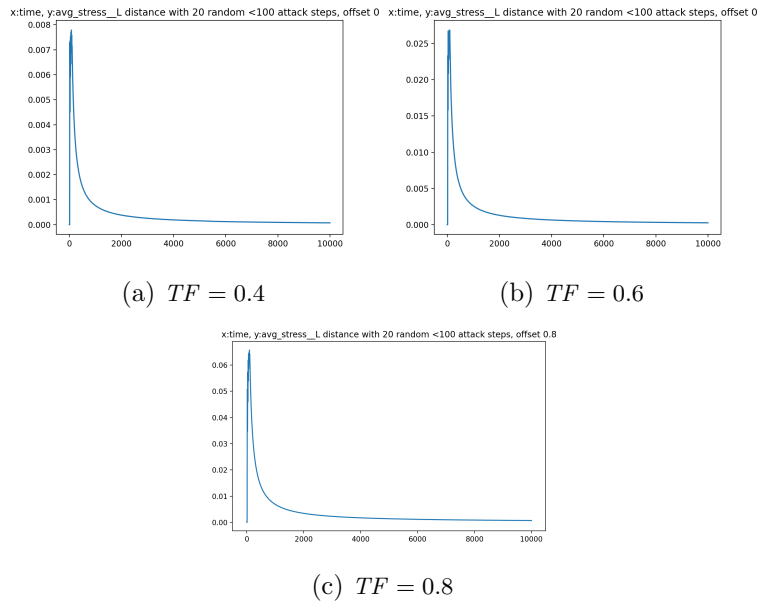


Figure 19. Estimation of average stress distance between a genuine engine and an engine subject to tampering with sensor $temp_L$. The simulation settings are the same of Figure 18.

variables, one counting the number of instants with the stress condition and one counting the total number of steps. Robustness can be estimated as in the earlier case. In Figure 19 we report the robustness of the left engine Eng_L . Here we use the same simulation settings used in Figure 18 and the same offset values. The figure shows that we have robustness for low values of η_2 , which, in essence, tells us that attacks in a given window are unable to generate overstress in the long run.

8. CONCLUDING REMARKS

The notion of robustness is used in different contexts, from control theory [ZD98] to biology [Kit07]. Commonly, it is meant as the ability of a system to maintain its functionalities against external and internal perturbations. In the context of CPSs, [FKP16] individuates five kinds of robustness: (i) input/output robustness; (ii) robustness with respect to system parameters; (iii) robustness in real-time system implementation; (iv) robustness due to unpredictable environment; (v) robustness to faults. The notions of adaptability and reliability considered in this paper fall in category (iv). In particular, in the example of the engine system the attacks are the source of environment's unpredictability.

We notice that our notions of robustness differ from classical ones like those in [FP09, DM10]. Our notions require to compare all behaviours of two different systems, whereas [FP09, DM10] compare a single behaviour of a single system with the set of the behaviours that satisfy a given property, which is specified by means of a formula expressed in a suitable temporal logic.

Moreover, the generality of our notions of robustness allows us to capture classical properties of linear systems. For instance, our notion of adaptability can be used to encode the overshoot [ZNTH19] and settling time [Dod08] of signals. Indeed, there are substantial differences between them. Specifically: (i) adaptability is a property of CPSs in general, whereas overshoot and settling time are referred to signals; (ii) adaptability is defined over the evolution sequence of a system, i.e., over all possible trajectories, whereas the other notions are dependency measures of a single signal; (iii) using the notion of Definition 6.3, the overshoot can be expressed as the parameter η_1 , and the settling time as $\tilde{\tau}$. This means that not only our notion of adaptability captures both properties, but it can be used to study their correlation.

As a first step for future research we will provide a simple logic, defined in the vein of *Signal Temporal Logic* (STL) [MN04], that can be used to specify requirements on the evolution sequences of a system. Our intuition is that we can exploit the evolution metric, and the algorithm we have proposed, to develop a quantitative model checking tool for this type of systems. Recently we have developed the *Robustness Temporal Logic* (*RobTL*) [CLT22], together with its model checker that is integrated in our *Software Tool for the Analysis of Robustness in the unKnown environment* (STARK) [CLT23]. This logic allows us to specify temporal requirements on the evolution of distances between the nominal behaviour of a system and its perturbed version, thus including properties of robustness against uncertainties of programs. However, we plan to propose a temporal logic allowing us to deal with the presence of uncertainties from another point of view: our aim is to express requirements on the *expected* behaviour of the system in the presence of uncertainties and perturbations, by using probability measures over data states as atomic propositions.

Moreover, we would like to enhance the modelling of time in our framework. Firstly we could relax the timing constraints on the evolution metric by introducing a *time tolerance* and defining a *stretched evolution metric* as a Skorokhod-like metric [Sko56], as those used for conformance testing [DMP17]. Then, we could provide an extension of our techniques to the case in which also the program shows a continuous time behaviour.

The use of metrics for the analysis of systems stems from [GJS90, DGJP04, KN96] where, in a process algebraic setting, it is argued that metrics are indeed more informative than behavioural equivalences when quantitative information on the behaviour is taken into account. The Wasserstein lifting has then found several successful applications: from the

definition of *behavioural metrics* (e.g., [vB05, CLT20b, GLT16, GT18]), to privacy [CGPX14, CCP18, CCP20] and machine learning (e.g., [ACB17, GAA⁺17, TBGS18]). Usually, one can use behavioural metrics to quantify how well an implementation (I) meets its specification (S). In [CHR12] the authors do so by setting a two players game with weighted choices, and the cost of the game is interpreted as the distance between I and S . Hence the authors propose three distance functions: *correctness*, *coverage*, and *robustness*. Correctness expresses how often I violates S , coverage is its dual, and robustness measures how often I can make an unexpected error with the resulting behaviour still meeting S . A similar game-based approach is used in [CDDP19] to define a *masking fault-tolerant* distance. Briefly, a system is masking fault-tolerant if faults do not induce any observable behaviour in the system. Hence, the proposed distance measures how many faults are tolerated by I while being masked by the states of the system. Notice that the notion of robustness from [CHR12] and the masking fault-tolerant distance from [CDDP19] are quite different from our notions of reliability and robustness. In fact, we are not interested in counting how many times an error occurs, but in checking whether the system is able to regain the desired behaviour after the occurrence of an error.

One of the main objectives in the control theory research area is the synthesis of controllers satisfying some desired property, such as safety, stability, robustness, etc. Conversely, our purpose in this paper was to provide some tools for the *analysis* of the interaction of a *given* program with an environment. The idea is that these tools can be used to test a synthesised controller against its deployment in various environments. While a direct application of our framework to the synthesis of programs does not seem feasible, it would be interesting to investigate whether a combination of it with learning techniques can be used for the design and synthesis of robust controllers. Our confidence in a potential application relies on some recently proposed metric-based approaches to controllers synthesis [GBS⁺19, AP11], in which, however, the environment is only modelled deterministically.

To conclude, we remark that our evolution sequences are not a rewriting of Markov processes as *transformers of distributions* [KA04, KVAK10]. Roughly, in the latter approach, one can interpret state-to-state transition probabilities as a single distribution over the state space, so that the behaviour of the system is given by the sequence of the so obtained distributions. While the two approaches may seem similar, there are some substantial differences. Firstly, the state space in [KA04, KVAK10] is finite and discrete, whereas here we are in the continuous setting (cf. Remark 1.1). Secondly, the transformers of distributions consider the behaviour of the system as a whole, i.e., it is not possible to separate the logical component from the environment.

ACKNOWLEDGEMENTS

This work is supported by the project *Programs in the wild: uncertainties, adaptability and verification* (ULTRON) of the Icelandic Research Fund (grant No. 228376-051). Moreover, V. Castiglioni has been supported by the project *Open Problems in the Equational Logic of Processes* (OPEL) of the Icelandic Research Fund (grant No. 196050-051).

REFERENCES

- [ACB17] Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of ICML 2017*, pages 214–223, 2017. URL: <http://proceedings.mlr.press/v70/arjovsky17a.html>.
- [ADB11] Alessandro Abate, Alessandro D’Innocenzo, and Maria Domenica Di Benedetto. Approximate abstractions of stochastic hybrid systems. *IEEE Trans. Automat. Contr.*, 56(11):2688–2694, 2011. doi:10.1109/TAC.2011.2160595.
- [AGA⁺17] Daniele Antonioli, Hamid Reza Ghaeini, Sridhar Adepur, Martín Ochoa, and Nils Ole Tippenhauer. Gamifying ICS security training and research: Design, implementation, and results of S3. In *Proceedings of CPS-SPC@CCS 2017*, pages 93–102. ACM, 2017. doi:10.1145/3140241.3140253.
- [AKLP10] Alessandro Abate, Joost-Pieter Katoen, John Lygeros, and Maria Prandini. Approximate model checking of stochastic hybrid systems. *Eur. J. Control*, 16(6):624–641, 2010. doi:10.3166/ejc.16.624-641.
- [AL92] Karl J. Astrom and Michael Lundh. Lund control program combines theory with hands-on experience. *IEEE control systems*, 3(12):22–30, 1992. doi:10.1109/37.165511.
- [ALGG⁺06] Ignacio Alvarado, Daniel Limon, Winston García-Gabín, Teodoro Alamo, and Eduardo F. Camacho. An educational plant based on the quadruple-tank process. *IFAC Proceedings Volumes*, 39(6):82–87, 2006. 7th IFAC Symposium on Advances in Control Education. doi:10.3182/20060621-3-ES-2905.00016.
- [AP11] Alessandro Abate and Maria Prandini. Approximate abstractions of stochastic systems: A randomized method. In *Proceedings of CDC-ECC 2011*, pages 4861–4866, 2011. doi:10.1109/CDC.2011.6161148.
- [BNL13] Marco Bernardo, Rocco De Nicola, and Michele Loreti. A uniform framework for modeling nondeterministic, probabilistic, stochastic, or mixed processes and their behavioral equivalences. *Inf. Comput.*, 225:29–82, 2013. doi:10.1016/j.ic.2013.02.004.
- [Bog07] Vladimir I. Bogachev. *Measure Theory*. Number vol. 2 in Measure Theory. Springer-Verlag, Berlin/Heidelberg, 2007. doi:10.1007/978-3-540-34514-5.
- [CCP18] Valentina Castiglioni, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. A logical characterization of differential privacy via behavioral metrics. In *Proceedings of FACS 2018*, volume 11222 of *Lecture Notes in Computer Science*, pages 75–96, 2018. doi:10.1007/978-3-030-02146-7_4.
- [CCP20] Valentina Castiglioni, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. A logical characterization of differential privacy. *Sci. Comput. Program.*, 188:102388, 2020. doi:10.1016/j.scico.2019.102388.
- [CDDP19] Pablo F. Castro, Pedro R. D’Argenio, Ramiro Demasi, and Luciano Putruele. Measuring masking fault-tolerance. In *Proceedings of TACAS 2019, Part II*, volume 11428 of *Lecture Notes in Computer Science*, pages 375–392, 2019. doi:10.1007/978-3-030-17465-1_21.
- [CGPX14] Konstantinos Chatzikokolakis, Daniel Gebler, Catuscia Palamidessi, and Lili Xu. Generalized bisimulation metrics. In *Proceedings of CONCUR 2014*, pages 32–46, 2014. doi:10.1007/978-3-662-44584-6_4.
- [CH08] Federica Ciocchetta and Jane Hillston. Bio-pepa: An extension of the process algebra PEPA for biochemical networks. *Electron. Notes Theor. Comput. Sci.*, 194(3):103–117, 2008. doi:10.1016/j.entcs.2007.12.008.
- [CHR12] Pavol Cerný, Thomas A. Henzinger, and Arjun Radhakrishna. Simulation distances. *Theor. Comput. Sci.*, 413(1):21–35, 2012. doi:10.1016/j.tcs.2011.08.002.
- [CLe07] Christos G. Cassandras, John Lygeros, and (eds.). *Stochastic Hybrid Systems*. Number 24 in Control Engineering. CRC Press, Boca Raton, 1st edition, 2007. doi:10.1201/9781315221625.
- [CLT20a] Valentina Castiglioni, Michele Loreti, and Simone Tini. Measuring adaptability and reliability of large scale systems. In *Proceedings of ISoLA 2020*, volume 12477 of *Lecture Notes in Computer Science*, pages 380–396, 2020. doi:10.1007/978-3-030-61470-6_23.
- [CLT20b] Valentina Castiglioni, Michele Loreti, and Simone Tini. The metric linear-time branching-time spectrum on nondeterministic probabilistic processes. *Theor. Comput. Sci.*, 813:20–69, 2020. doi:10.1016/j.tcs.2019.09.019.

- [CLT21] Valentina Castiglioni, Michele Loreti, and Simone Tini. How adaptive and reliable is your program? In *Proceedings of FORTE 2021*, volume 12719 of *Lecture Notes in Computer Science*, pages 60–79. Springer, 2021. doi:10.1007/978-3-030-78089-0_4.
- [CLT22] Valentina Castiglioni, Michele Loreti, and Simone Tini. RobTL: A temporal logic for the robustness of cyber-physical systems. *CoRR*, abs/2212.11158, 2022. arXiv:2212.11158, doi:10.48550/arXiv.2212.11158.
- [CLT23] Valentina Castiglioni, Michele Loreti, and Simone Tini. STARK: A software tool for the analysis of robustness in the unknown environment. In *Proceedings of COORDINATION 2023*, volume 13908 of *Lecture Notes in Computer Science*, 2023. To appear. doi:10.1007/978-3-031-35361-1_6.
- [dAHM03] Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. Discounting the Future in Systems Theory. In *Proceedings of ICALP'03*, ICALP '03, pages 1022–1037. Springer, 2003. doi:10.1007/3-540-45061-0_79.
- [DGJP04] Josee Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled Markov processes. *Theor. Comput. Sci.*, 318(3):323–354, 2004. doi:10.1016/j.tcs.2003.09.013.
- [DM10] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In Krishnendu Chatterjee and Thomas A. Henzinger, editors, *Proceedings of FORMATS 2010*, volume 6246 of *Lecture Notes in Computer Science*, pages 92–106. Springer, 2010. doi:10.1007/978-3-642-15297-9_9.
- [DMP17] Jyotirmoy V. Deshmukh, Rupak Majumdar, and Vinayak S. Prabhu. Quantifying conformance using the skorokhod metric. *Formal Methods in System Design*, 50(2-3):168–206, 2017. doi:10.1007/s10703-016-0261-8.
- [Dod08] Stephen J. Dodds. Settling time formulae for the design of control systems with linear closed loop dynamics. In *Proceedings of Advances in Computing and Technology 2008*, pages 31–39, 2008. URL: <https://repository.uel.ac.uk/item/86590>.
- [FKP16] Martin Fränzle, James Kapinski, and Pavithra Prabhakar. Robustness in cyber-physical systems. *Dagstuhl Reports*, 6(9):29–45, 2016. doi:10.4230/DagRep.6.9.29.
- [FP09] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.*, 410(42):4262–4291, 2009. doi:10.1016/j.tcs.2009.06.021.
- [FR18] Olivier P. Faugeras and Ludeger Rüschendorf. Risk excess measures induced by hemi-metrics. *Probability, Uncertainty and Quantitative Risk*, 3:6, 2018. doi:10.1186/s41546-018-0032-0.
- [GAA⁺17] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of Wasserstein GANs. In *Proceedings of Advances in Neural Information Processing Systems*, pages 5767–5777, 2017. URL: <http://papers.nips.cc/paper/7159-improved-training-of-wasserstein-gans>.
- [GBS⁺19] Shromona Ghosh, Somil Bansal, Alberto L. Sangiovanni-Vincentelli, Sanjit A. Seshia, and Claire Tomlin. A new simulation metric to determine safe environments and controllers for systems with unknown dynamics. In *Proceedings of HSCC 2019*, pages 185–196, 2019. doi:10.1145/3302504.3311795.
- [GGI⁺15] Dieter Gollmann, Pavel Gurikov, Alexander Isakov, Marina Krotofil, Jason Larsen, and Alexander Winnicki. Cyber-Physical Systems Security: Experimental Analysis of a Vinyl Acetate Monomer Plant. In *Proceedings of CPSS 2015*, pages 1–12. ACM, 2015. doi:10.1145/2732198.2732208.
- [GJS90] Alessandro Giacalone, Chi-Chang Jou, and Scott A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proceedings of IFIP Work, Conf. on Programming, Concepts and Methods*, pages 443–458, 1990.
- [GLT16] Daniel Gebler, Kim G. Larsen, and Simone Tini. Compositional bisimulation metric reasoning with probabilistic process calculi. *Log. Methods Comput. Sci.*, 12(4), 2016. doi:10.2168/LMCS-12(4:12)2016.
- [GT18] Daniel Gebler and Simone Tini. SOS specifications for uniformly continuous operators. *J. Comput. Syst. Sci.*, 92:113–151, 2018. doi:10.1016/j.jcss.2017.09.011.
- [HHH⁺94] Jane Hillston, Holger Hermanns, Ulrich Herzog, Vassilis Mertsiotakis, and Michael Rettelbach. Stochastic process algebras: integrating qualitative and quantitative modelling. In *Proceedings of FORTE 1994*, volume 6 of *IFIP Conference Proceedings*, pages 449–451, 1994.

- [HJS⁺14] Guillermo Heredia, A. E. Jimenez-Cano, I. Sánchez, Domingo Llorente, V. Vega, J. Braga, José Ángel Acosta, and Aníbal Ollero. Control of a multirotor outdoor aerial manipulator. In *Proceedings of IROS 2014*, pages 3417–3422. IEEE, 2014. doi:10.1109/IROS.2014.6943038.
- [HLS00] Jianghai Hu, John Lygeros, and Shankar Sastry. Towards a theory of stochastic hybrid systems. In *Proceedings of HSCC 2000*, volume 1790 of *Lecture Notes in Computer Science*, pages 160–173, 2000. doi:10.1007/3-540-46430-1_16.
- [Joh00] Karl H. Johansson. The quadruple-tank process: a multivariable laboratory process with an adjustable zero. *IEEE Trans. Control. Syst. Technol.*, 8(3):456–465, 2000. doi:10.1109/87.845876.
- [KA04] YoungMin Kwon and Gul Agha. Linear inequality LTL (iltl): A model checker for discrete time Markov chains. In *Proceedings of ICFEM 2004*, volume 3308 of *Lecture Notes in Computer Science*, pages 194–208, 2004. doi:10.1007/978-3-540-30482-1_21.
- [Kit07] Hiroaki Kitano. Towards a theory of biological robustness. *Molecular Systems Biology*, 3(1):137, 2007. doi:10.1038/msb4100179.
- [KN96] Marta Z. Kwiatkowska and Gethin Norman. Probabilistic metric semantics for a simple language with recursion. In *Proceedings of MFCS’96*, volume 1113 of *Lecture Notes in Computer Science*, pages 419–430, 1996. doi:10.1007/3-540-61550-4_167.
- [Kop11] Hermann Kopetz. *Internet of Things*, pages 307–323. Springer US, Boston, MA, 2011. doi:10.1007/978-1-4419-8237-7_13.
- [KVAK10] Vijay Anand Korthikanti, Mahesh Viswanathan, Gul Agha, and YoungMin Kwon. Reasoning about MDPs as transformers of probability distributions. In *Proceedings of QEST 2010*, pages 199–208. IEEE Computer Society, 2010. doi:10.1109/QEST.2010.35.
- [LMMT21] Ruggero Lanotte, Massimo Merro, Andrei Munteanu, and Simone Tini. Formal impact metrics for cyber-physical attacks. In *Proceedings of CSF 2021*, pages 1–16, 2021. doi:10.1109/CSF51468.2021.00040.
- [MN04] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Proceedings of FORMATS and FTRTFT 2004*, volume 3253 of *Lecture Notes in Computer Science*, pages 152–166, 2004. doi:10.1007/978-3-540-30206-3_12.
- [MT16] Aditya P. Mathur and Nils Ole Tippenhauer. Swat: a water treatment testbed for research and training on ICS security. In *Proceedings of CySWater@CPSWeek 2016*, pages 31–36. IEEE Computer Society, 2016. doi:10.1109/CySWater.2016.7469060.
- [Put05] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, United States, 2005.
- [RKO⁺97] Jörg Raisch, Eberhard Klein, Siu O’Young, Christian Meder, and Alexander Itigin. Approximating automata and discrete control for continuous systems - two examples from process control. In *Proceedings of Hybrid Systems V, 1997*, volume 1567 of *Lecture Notes in Computer Science*, pages 279–303, 1997. doi:10.1007/3-540-49163-5_16.
- [RKSF13] Svetlozar T. Rachev, Lev B. Klebanov, Stoyan V. Stoyanov, and Frank J. Fabozzi. *The Methods of Distances in the Theory of Probability and Statistics*. Springer, 2013.
- [RLSS10] Ragunathan Rajkumar, Insup Lee, Lui Sha, and John A. Stankovic. Cyber-physical systems: the next computing revolution. In *Proceedings of DAC 2010*, pages 731–736. ACM, 2010. doi:10.1145/1837274.1837461.
- [Roy88] Halsey L. Royden. *Real Analysis*. Macmillan, New York, NY, 3rd edition, 1988.
- [Seg95] Roberto Segala. *Modeling and verification of randomized distributed real-time systems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1995. URL: <http://hdl.handle.net/1721.1/36560>.
- [SFG⁺21] Bharath K. Sriperumbudur, Kenji Fukumizu, Arthur Gretton, Bernard Schölkopf, and Gert R. G. Lanckriet. On the empirical estimation of integral probability metrics. *Electronic Journal of Statistics*, 6:1550–1599, 2021. doi:10.1214/12-EJS722.
- [Sko56] Anatoliy V. Skorokhod. Limit theorems for stochastic processes. *Theory Probab. Appl.*, 1:261–290, 1956.
- [TBGS18] Ilya O. Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schölkopf. Wasserstein auto-encoders. In *Proceedings of ICLR 2018*, 2018. URL: <https://openreview.net/forum?id=HkL7n1-0b>.

- [TK10] David Thorsley and Eric Klavins. Approximating stochastic biochemical processes with Wasserstein pseudometrics. *JET Syst. Biol.*, 4(3):193–211, 2010. doi:10.1049/iet-syb.2009.0039.
- [Val74] S. S. Vallender. Calculation of the Wasserstein distance between probability distributions on the line. *Theory Probab. Appl.*, 18(4):784–786, 1974.
- [Vas69] Leonid N. Vaserstein. Markovian processes on countable space product describing large systems of automata. *Probl. Peredachi Inf.*, 5(3):64–72, 1969.
- [vB05] Franck van Breugel. A behavioural pseudometric for metric labelled transition systems. In *Proceedings of CONCUR 2005*, pages 141–155, 2005. doi:10.1007/11539452_14.
- [vGSS95] Rob J. van Glabbeek, Scott A. Smolka, and Bernhard Steffen. Reactive, generative and stratified models of probabilistic processes. *Inf. Comput.*, 121(1):59–80, 1995. doi:10.1006/inco.1995.1123.
- [Vil08] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer, 2008.
- [VNGV16] Csaba Virágh, Máté Nagy, Carlos Gershenson, and Gábor Vásárhelyi. Self-organized UAV traffic in realistic environments. In *Proceedings of IROS 2016*, pages 1645–1652. IEEE, 2016. doi:10.1109/IROS.2016.7759265.
- [ZD98] K. Zhou and J. C. Doyle. *Essentials of Robust Control*. Prentice-Hall, 1998.
- [ZNTH19] Guanglei Zhao, Dragan Nešić, Ying Tan, and Changchun Hua. Overcoming overshoot performance limitations of linear systems with reset control. *Automatica*, 101:27–35, 2019. doi:10.1016/j.automatica.2018.11.038.