Online-Monitoring Autonomous Transport Robots with an R-valued Temporal Logic

Felix Lorenz
Technical University Berlin and *Fraunhofer FOKUS*
Berlin, Germany

Holger Schlingloff
Humboldt University and *Fraunhofer FOKUS*
Berlin, Germany

***Abstract*—In this paper, we introduce real-valued temporal logic (RVTL) for online monitoring of reactive and cyber-physical systems. Our approach is based on classical metric temporal logic (MTL) with a real-valued semantics, where the truth value of a formula with respect to a finite trace depends on the distance between the end of the trace and the bound of the temporal operators in the formula. The assumed time model is dense and pointwise, i.e., the basic propositions refer to events reported by the system at specific times. We show that our logic is applicable for collaborating cyber-physical systems by giving example formulae from a case study of autonomous transport robots in a factory. We sketch an algorithm for monitoring RVTL formulae at runtime, and report on experiences with this algorithm in an actual industrial deployment of the case study.**

***Index Terms*—Cyber-physical Production Systems and Industry 4.0, Industrial Robots, Factory Automation**

I. Introduction

Online monitoring is a lightweight formal method, where the execution of a system is compared to its specification at runtime. One challenge in online monitoring is to define a suitable formalism that is expressive, easy to use and allows efficient monitoring algorithms. A well-established specification language for monitoring is Metric Temporal Logic (MTL) [1]. It extends the usual temporal operators "always", "sometime" and "until" with quantitative timing constraints. However, a problem with the usual semantics of MTL is that the verdicts can only assume the usual boolean truth values "true" and "false". Thus, at runtime, exceptional system behavior can be detected only upon specification violation, possibly too late to initiate appropriate corrective actions. In [2], we defined a five-valued logic for runtime monitoring based on Linear Temporal Logic (LTL). This approach can deal with incomplete information, arising from the fact that during online monitoring of an interactive system only part of the observed trace is available at any given instant. Furthermore, it can handle uncertainties in monitoring arising from the fact that observations in a distributed system are possibly inaccurate due to interleaving and internal nondeterminism of the components. However, as LTL does not incorporate real-time constructs, that approach can monitor timing properties only indirectly.

Here, we use a fragment of MTL instead, which has temporal operators indexed by a real number to express deadlines. We replace the usual semantics of MTL with a real-valued semantics, where every formula is evaluated according to the distance to a deadline. This way, more information about the system state is available and decisions about corrective actions can be made earlier than with the usual two-valued MTL semantics. Particularly in large fleets with many concurrent processes, runtime faults can propagate and cause failures in other components, which can only be detected with real-time information about the system group state. We describe our semantics with a few example formulas, and report on the application to an industrial use case.

Our paper is structured as follows: Following this introduction, in Section II we discuss real-time logics for monitoring in general and MTL in particular. Subsequently, in Section III, we describe our case study of collaborative autonomous transport robots in a production factory environment. Then, in Section IV, we define our logic for online-monitoring, which we call R-Valued Temporal Logic (RVTL). In Section V, we give some results of evaluating RVTL formulas on large traces obtained from our case study. Finally, in Section VI, we conclude our contribution by giving some directions for further work.

II. Related Work

Online monitoring as a particular form of runtime verification (RV) is a subject which emerged already in the 1970's, but has received increased attention lately. For example, Ahrendt et al. [3] report on the combination of static and runtime verification for online monitoring of a open-source Java shopping cart web application. Much of the work in RV has been driven by the development of practical tools. Already in 2004, Delgado, Gates and Roach [4] gave a survey comparing different runtime software-fault monitoring tools available at the time. In 2014, the first international competition on software for runtime verification took place ([5], [6]). Nine different tools competed in specification and online monitoring of C and Java programs. Although the results were quite impressive with respect to memory footprint and runtime overhead, the authors conclude that "designing a monitoring framework that is portable, robust in its safety guarantees, and minimally expensive remains an open problem."

Specification formalisms for runtime verification often are based on classical automata theory or similar modelling notations. For example, Delahaye, Kosmatov, and Signoles [7] define an automaton-based specification language for monitoring and analyzing C programs. d'Amorim and Rosu [8]

use a monitoring language based on ω-automata. In [9], we defined a monitoring specification language as an extension of the live sequence charts defined by Harel and Damm. We showed that this language has expressive power comparable to linear temporal logic and ω-regular languages.

Metric temporal logic (MTL) emerged in the 1970's as a real-time extension to linear temporal logic. Koymans [1] described its application to the specification of quantitative temporal properties. It is a popular formalism to specify the expected ordering and timing between events in reactive and hybrid systems [10], and has been thoroughly analyzed in terms of expressiveness [11], [12] and complexity [13], [14]. Ho, Ouaknine and Worrell [15] described the application of MTL to online monitoring. Their algorithm is based on rewriting arbitrary MTL formulas into LTL formulas to obtain a good complexity. In contrast, we focus on the early detection of potential problems. In [16], the authors defined a "robust" semantics for MTL which inspired our R-valued semantics. With robust semantics, the truth value of a formula over a finite trace serves as an estimate of how far the execution trajectory is from satisfying the respective property. However, all prior works using this or similar robust semantics rely on the metric distance of a signal to its specification for a robustness estimate, whereas for RVTL we use the temporal distance to a deadline. Additionally, by imposing specific syntactical restrictions upon the temporal operators, we obtain specification parameters that correspond to intuitively understandable concepts: *Deadlines* and *timeouts*.

III. Collaborating Transport Robots

Modern workflows as envisioned and encountered in industry 4.0 production plants are characterized by highly automated processes, a frequently changing arrangement of assembly units and autonomous subcomponents. In such a setting, transport robots will replace conveyor belts for more seamless integration and scaling with a dynamically changing factory layout.

In order for the production plant to achieve its intended output, *all transport orders must be completed within a certain amount of time*. Otherwise the machines will eventually halt due to congestion or shortage of supplies. This constitutes the main goal of a Collaborating Autonomous Transport Robot (CATR) fleet, subordinate to which the robots have individual goals such as finding the shortest path to the goal locations or not depleting their battery. While in legacy management architectures a single orchestration unit operated the whole fleet, the transition to autonomous behavior necessitates the addition of negotiation and consensus protocols.

In our example, the operating procedure of a CATR fleet is as follows: After a machine has announced a pending transport job to the broadcast network, the robots place bids and agree on a winner akin to a consensus-based auction algorithm such as the one introduced in [17]. The winning robot adds the job to its queue and moves to the source machine, subject to dedicated path and goal conflict resolution protocols. Upon arrival, the robot obtains the load via docking and load exchange procedures. Later the navigation and load transfer steps are repeated for the delivery part of the order. Finally, the robot either starts the next job in its queue or finds and moves to an empty parking spot.

Examining the execution logs of an actual fleet of nine transport robots (legacy architecture), we need to determine the boundary between expected and exceptional runtime behaviour in order to define the scope of the monitoring problem. The log file spans five full days and consists of roughly 700,000 steps at a rate of 1.3 events per second. The logged events can be assigned to one of five categories, as presented in Figure 1. The vast majority of log messages (more than 665K) consisted of robots reporting their current position. Among the robots in the system, the events and event types were almost uniformly distributed.

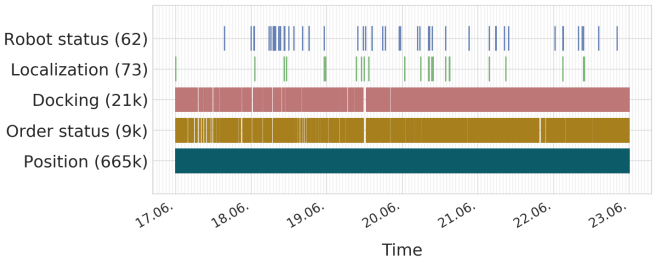

Fig. 1: Distribution of events over time in the logged data.

The distributions show that robots being reported offline and localizations frequently co-occur, hinting at a common physical failure source. Since the monitor is instrumented to obtain its trace from overhearing the broadcast network (*bus-monitor architecture*, c.f. [18]), it can not directly detect faults beyond those an individual robot can detect itself. Instead we have to monitor for bounded reactiveness, safety and timeout violations that may ultimately lead to non-fulfillment of the global goal of the fleet.

Figure 2 shows the durations of three essential processes in the fleet's standard operating procedure, as contained in the log. Here, "order processing" refers to the full job duration, from order assignment to successful load delivery.

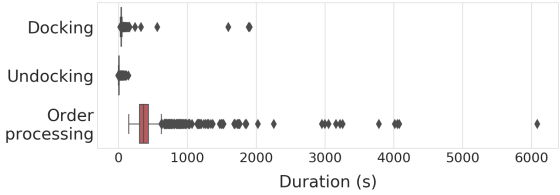

Fig. 2: Measured durations in the industrial use case

In the observed trace, undocking maneuvers have very regular durations, while for order processing and docking there are are several outliers where the process takes much longer than usual. An accumulation of such local problems could significantly affect the whole system. Subsequently, we

will show how to use quantitative temporal constraints to detect such exceptions by monitoring.

IV. R-valued Metric Temporal Logic

Before we introduce our method, we establish a few preliminary concepts that will be needed later on. We assume a pointwise time model, where the System Under Monitoring (SUM) produces a timestamped sequence of instantenous events.

Definition 1 (Trace)**.** *A trace $\rho \in (\mathbb{R}^{\geq 0} \times \Sigma)^*$ is a non-empty, finite sequence of tuples $(\tau_1, \sigma_1), (\tau_2, \sigma_2), \ldots, (\tau_n, \sigma_n)$, where the events $\sigma_i \in \Sigma$ are taken from the finite alphabet $\Sigma = 2^{\mathbb{P}}$ and the timestamps $\tau_i \in \mathbb{R}^{\geq 0}$ form a time sequence as follows: $\tau_1 = 0$ and $\tau_i \leq \tau_{i+1}$ for all i with $1 \leq i \leq |\tau|$.*

Given a formula φ in some temporal logic, we say that a trace ρ is a model of φ if $(\rho, i) \vDash \varphi$ for all $1 \leq i \leq |\rho|$. The set of all models of φ is called the *language* of φ: $\mathcal{L}_\varphi = \{\rho \in (\mathbb{R}^{\geq 0} \times \Sigma)^* | (\rho, i) \vDash \varphi, 1 \leq i \leq |\rho|\}$.

Definition 2 (Good/bad prefix)**.** *Given a language $\mathcal{L}_\varphi \subseteq (\mathbb{R}^{\geq 0} \times \Sigma)^*$, we call a trace $\rho \in (\mathbb{R}^{\geq 0} \times \Sigma)^*$*

- *a bad prefix for $\mathcal{L}_\varphi$, if for all $w \in (\mathbb{R}^{\geq 0} \times \Sigma)^*$, $\rho \circ w \notin \mathcal{L}_\varphi$*
- *a good prefix for $\mathcal{L}_\varphi$, if for all $w \in (\mathbb{R}^{\geq 0} \times \Sigma)^*$, $\rho \circ w \in \mathcal{L}_\varphi$*

where $\circ$ is the concatenation between two traces.

We shall henceforth use the short notation $\rho \in \mathrm{bad}(\varphi)$ if ρ is a bad prefix of $\mathcal{L}_\varphi$ and $\rho \in \mathrm{good}(\varphi)$ respectively.

The functional and non-functional requirements of a CATR fleet necessitate the expression of quantitative timing constraints. More precisely, the constraints are intended to represent *deadlines* by which we expect the system to have or have not exerted certain behavior. We therefore restrict classical MTL such that intervals are always left-bounded by 0. Furthermore, we assume that intervals are right-bounded by a finite positive real number. Excluding unconstrained temporal operators from the language ensures monitorability: Specifications that exclusively contain bounded intervals are essentially safety properties [19]. Safety properties are monitorable at runtime [13].

This restricted version of MTL has all the required characteristics for efficient CATR monitoring; together with the $\overline{\mathbb{R}}$-valued semantics given below we call it RVTL.

Definition 3 (RVTL syntax)**.** *Given a finite set of atomic propositions $\mathbb{P}$, the syntax of a RVTL formula is defined by*

$$\varphi \coloneqq \bot \mid p \mid (\varphi \to \varphi) \mid (\varphi \; \mathcal{U}_t \; \varphi)$$

where $p \in \mathbb{P}$, and $t \in \mathbb{R}^{>0}$ is a positive, non-zero real number.

Other operators can be defined as usual: $\neg\varphi \coloneqq (\varphi \to \bot)$, $\top \coloneqq \neg\bot$, $(\varphi \vee \psi) \coloneqq (\neg\varphi \to \psi)$, $(\varphi \wedge \psi) \coloneqq \neg(\neg\varphi \vee \neg\psi)$, $\Diamond_t \; \varphi \coloneqq \top \; \mathcal{U}_t \; \varphi$, $\Box_t \; \varphi \coloneqq \neg\Diamond_t \; \neg\varphi$, etc.

Let $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$ and $(\overline{\mathbb{R}}, \leq)$ be its closure with the usual ordering relation. Further let $\sqcup : \overline{\mathbb{R}} \times \overline{\mathbb{R}} \to \overline{\mathbb{R}}$ and $\sqcap : \overline{\mathbb{R}} \times \overline{\mathbb{R}} \to \overline{\mathbb{R}}$ be the maximum and minimum functions on the extended domain, i.e., $(x \sqcup \infty) = (\infty \sqcup y) = \infty$, $(x \sqcup -\infty) = (-\infty \sqcup x) = x$, $(x \sqcap -\infty) = (-\infty \sqcap y) = -\infty$, $(x \sqcap \infty) = (\infty \sqcap x) = x$ and for $x, y \notin \{\infty, -\infty\}$, we have $(x \sqcup y) = \max(x, y)$ and $(x \sqcap y) = \min(x, y)$. Furthermore, for some subset $X \subseteq \overline{\mathbb{R}}$ let $\bigsqcap$ and $\bigsqcup$ be the supremum and infimum functions over the set X, with $\bigsqcup \overline{\mathbb{R}} = \infty$ and $\bigsqcap \overline{\mathbb{R}} = -\infty$. The robust semantics is defined using an evaluation function $[\![\varphi]\!](\rho, i) : ((\mathbb{R}^{\geq 0} \times \Sigma)^*, \mathbb{N}) \to \overline{\mathbb{R}}$ instead of the satisfaction notation $(\rho, i) \vDash \varphi$ used for qualitative semantics.

Definition 4 (RVTL semantics)**.** *The evaluation $[\![\varphi]\!](\rho, i)$ of a formula φ at step i (at time τ_i) of the trace ρ is defined inductively by*

$$[\![\bot]\!](\rho, i) = -\infty$$

$$[\![p]\!](\rho, i) = \begin{cases} \infty, & \text{if } p \in \sigma_i \\ -\infty, & \text{otherwise} \end{cases}$$

$$[\![(\varphi \to \psi)]\!](\rho, i) = (-[\![\varphi]\!](\rho, i) \sqcup [\![\psi]\!](\rho, i))$$

$$[\![(\varphi \; \mathcal{U}_t \; \psi)]\!](\rho, i) =$$

$$\begin{cases} (\tau_i + t - \tau_{|\rho|}), & \text{if } (\tau_i + t) \geq \tau_{|\rho|} \text{ and for all } k \text{ with} \\ & i \leq k \leq |\rho| \;.\; ([\![\varphi]\!](\rho, k) \neq -\infty \wedge [\![\psi]\!](\rho, k) \neq \infty) \\ \bigsqcup_{j \in \{l \in \mathbb{N} | (\tau_l - \tau_i) \leq t\}} \left([\![\psi]\!](\rho, j) \sqcap \bigsqcap_{i \leq k < j} [\![\varphi]\!](\rho, l) \right), \\ \text{otherwise.} \end{cases}$$

For the derived operators, we have:

$$[\![\neg\varphi]\!](\rho, i) = -[\![\varphi]\!](\rho, i)$$

$$[\![\varphi \vee \psi]\!](\rho, i) = [\![\varphi]\!](\rho, i) \sqcup [\![\psi]\!](\rho, i)$$

$$[\![\varphi \wedge \psi]\!](\rho, i) = [\![\varphi]\!](\rho, i) \sqcap [\![\psi]\!](\rho, i)$$

$$[\![\Diamond_t \; \varphi]\!](\rho, i) =$$

$$\begin{cases} (\tau_i + t - \tau_{|\rho|}), & \text{if } (\tau_i + t) \geq \tau_{|\rho|} \text{ and} \\ & [\![\varphi]\!](\rho, k) \neq \infty \text{ for all } k \text{ with } i \leq k \leq |\rho| \\ \bigsqcup_{j \in \{l \in \mathbb{N} | (\tau_l - \tau_i) \leq t\}} [\![\varphi]\!](\rho, j), & \text{otherwise.} \end{cases}$$

In particular, $[\![\Diamond_t \; \varphi]\!](\rho, i) = \infty$ iff there is a j with $i \leq j \leq |\rho|$ such that $(\tau_i + t) \geq \tau_j$ and $[\![\varphi]\!](\rho, j) = \infty$.

$$[\![\Box_t \; \varphi]\!](\rho, i) =$$

$$\begin{cases} -(\tau_i + t - \tau_{|\rho|}), & \text{if } (\tau_i + t) \geq \tau_{|\rho|} \text{ and} \\ & [\![\varphi]\!](\rho, k) \neq -\infty \text{ for all } k \text{ with } i \leq k \leq |\rho| \\ -\bigsqcup_{j \in \{l \in \mathbb{N} | (\tau_l - \tau_i) \leq t\}} -[\![\varphi]\!](\rho, j), & \text{otherwise.} \end{cases}$$

Intuitively, the semantics can be summarized as follows. The two boolean truth values correspond to $\pm\infty$. An evaluation result from the reals indicates that there are some continuations of ρ that satisfy the formula and some that don't (see Lemma 3). In that case, the value quantifies how much time is left for a certain event to (not) occur and decide the boolean truth value of the formula before its deadline is surpassed, similarly to the *robustness degree* of [16]. Example 1 illustrates the temporal evolution of the verdict $[\![\varphi]\!](\rho, 1)$ for three different formulae as the trace is observed event by event. The dashed lines indicate an evaluation to ∞ (green) or $-\infty$ (red).

Example 1. *Given the trace $\rho = ((0,\ a),\ (2,\ a),\ (4,\ b),\ (6,\ b),\ (8,\ b),\ (10,\ c),\ (12,\ c))$, consider two policy specifications:*

$$\varphi_1 = \Diamond_{15}\ c \qquad \textit{(Bounded response)}$$
$$\varphi_2 = \Box_{10}\ (a \vee b) \qquad \textit{(Bounded safety)}$$

Figure 3 presents the result of evaluating $[\![\varphi]\!](\rho_i, 1)$ for every prefix $\rho_i = ((\tau_1, \sigma_1), \ldots, (\tau_i, \sigma_i)), 1 \leq i \leq |\rho|$ of the trace ρ.

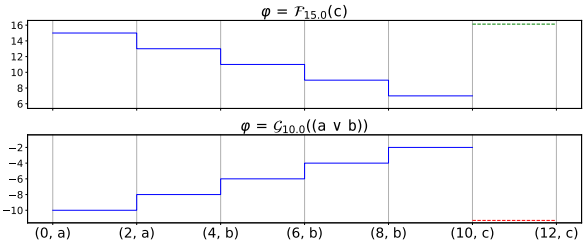

Fig. 3: Evaluation of $[\![\varphi_{1,2}]\!](\rho_i, 1)$ over every prefix ρ_i of the example trace.

For bounded response patterns like φ_1, the robustness value is initially high and positive and decreases towards zero with increasing prefix length. For bounded safety properties like φ_2, it starts negative and increases towards zero. At $\tau_6 = 10$ a c event occurs at which point $[\![\varphi_1]\!](\rho, 1) = \infty$ and $[\![\varphi_2]\!](\rho, 1) = -\infty$.

Next we discuss a series of key properties of RVTL. The following lemma states that a monitor following the RVTL semantics is *stable*, i.e., its verdict never changes from *true* (*false*) to something else upon reading the next event produced by the system.

Lemma 1. *Let φ be a RVTL formula and ρ be an arbitrary finite execution trace. Then for any $(\tau, \sigma) \in (\mathbb{R}^{\geq 0} \times \Sigma)$ it holds that*

$$[\![\varphi]\!](\rho, i) = \infty \Rightarrow [\![\varphi]\!](\rho \circ (\tau, \sigma), i) = \infty \qquad (1)$$
$$[\![\varphi]\!](\rho, i) = -\infty \Rightarrow [\![\varphi]\!](\rho \circ (\tau, \sigma), i) = -\infty \qquad (2)$$

Proof. We show stability by structural induction over φ. The base case where φ is composed of only symbols from $\mathbb{P} \cup \{\bot, \rightarrow\}$ trivially satisfies stability, because in that case $[\![\varphi]\!](\rho, i)$ only depends on σ_i which is fixed (Definition 1).

Case $\varphi = \varphi_1\ \mathcal{U}_t\ \varphi_2$: Assume that at time step $p = |\rho|$ with $\tau_p - \tau_i \leq t$ we have $[\![\varphi_2]\!](\rho, p) = \infty$, and for all q with $i \leq q < p$ it holds that $[\![\varphi_1]\!](\rho, q) = \infty$. Then $[\![\varphi]\!](\rho, i) = \bigsqcup_{j \leq p} \left([\![\varphi_2]\!](\rho, j) \sqcap \bigsqcap_{i \leq k < j} [\![\varphi_1]\!](\rho, l) \right) = \infty$, which is the least upper bound of $\overline{\mathbb{R}}$ and thus appending an arbitrary suffix to ρ will not change the result of the outer supremum.

Conversely, assume that at time step $p = |\rho|$ with $\tau_p - \tau_i \leq t$ we have $[\![\varphi_1]\!](\rho, p) = -\infty$, and for all q with $i \leq q \leq p$ we have $[\![\varphi_2]\!](\rho, q) = -\infty$. Then $\bigsqcup_{j \leq p} \left([\![\varphi_2]\!](\rho, j) \sqcap \bigsqcap_{i \leq k < j} [\![\varphi_1]\!](\rho, l) \right) = -\infty$ because for all q with $i \leq q \leq p$ it holds that $[\![\varphi_2]\!](\rho, q) = -\infty$ by assumption. Likewise for any $j > p$ it holds that $\bigsqcap_{i \leq k < j} [\![\varphi_1]\!](\rho, l) = -\infty$ because $[\![\varphi_1]\!](\rho, p) = -\infty$ by assumption. Finally, since $-\infty$ is the greatest lower bound of $\overline{\mathbb{R}}$, thus no addition to the set will change the result of the inner infimum. □

Next we show that for any given formula expressible in RVTL its good and bad prefixes coincide with their MTL counterparts.

Lemma 2. *Let φ be an arbitrary RVTL formula and $\widetilde{\varphi}$ be its counterpart expressed in MTL by replacing all constraints t with the corresponding intervals $[0, t]$. Further let $\rho \in (\mathbb{R}^{\geq 0} \times \Sigma)^*$ be an arbitrary finite trace. Then*

$$\rho \in \text{good}(\varphi) \iff \rho \in \text{good}(\widetilde{\varphi}) \qquad (3)$$
$$\rho \in \text{bad}(\varphi) \iff \rho \in \text{bad}(\widetilde{\varphi}) \qquad (4)$$

Proof. We prove our claim by structural induction on φ. If φ contains no temporal operators, every trace is a good or a bad prefix of φ and the equivalence immediately follows from the semantics of MTL and RVTL. To complete the induction we show that it also holds for the non-trivial case.

Case $\varphi = \varphi_1\ \mathcal{U}_t\ \varphi_2$ ($\widetilde{\varphi} = \widetilde{\varphi}_1\ \mathcal{U}_{[0,t]}\ \widetilde{\varphi}_2$). First we show $\rho \in \text{good}(\widetilde{\varphi}) \implies \rho \in \text{good}(\varphi)$. If $\rho \in \text{good}(\widetilde{\varphi})$ then there exists a time step p with $i \leq p \leq |\rho|$, $\tau_p \in [0, t]$, $(\rho, p) \vDash \widetilde{\varphi}_2$, and for all $i \leq q < p$ it holds that $(\rho, q) \vDash \widetilde{\varphi}_1$. Then by applying the base case we get $[\![\varphi]\!](\rho, p) = \infty$ and by Lemma 1 it follows that $[\![\varphi]\!](\rho \circ (\tau, \sigma), i) = \infty$ for any $(\tau, \sigma) \in (\mathbb{R}^{\geq 0} \times \Sigma)$. Hence $\rho \in \text{good}(\varphi)$.

To complete the equivalence, it remains to show that $\rho \in \text{good}(\varphi) \implies \rho \in \text{good}(\widetilde{\varphi})$. By Lemma 1 it is established that for $\rho \in \text{good}(\varphi)$, there must be a time step p with $i \leq p \leq |\rho|$, $\tau_p - \tau_i \leq t$, $[\![\varphi_2]\!](\rho, p) = \infty$, and for all $i \leq q < p$ it holds that $[\![\varphi_1]\!](\rho, q) = \infty$. Then the base case again gives $(\rho, i) \vDash \widetilde{\varphi}$ and by stability of MTL it is true that $\rho \in \text{good}(\widetilde{\varphi})$.

The proof for bad prefixes proceeds analogously. This completes the induction step. □

Algorithm An essential quality of an online monitoring algorithm is *trace-length independence*, that is, a monitor should not need to store the trace in its memory in order to produce verdicts. A common approach to achieve trace-length independence is to consume the events as they are received by the monitor and use them to update some internal state [20].

Our Algorithm for online monitoring of RVTL is based on storing and recursively updating subformulae upon the arrival of a new event. The main monitoring procedure `monitor`(φ, ρ) contains the instrumentation which directly observes the events as they are produced by the system group. For every event (τ_i, σ_i), it creates a formula object φ from a specification written in RVTL and replaces any atomic proposition p that does not occur within the scope of a temporal operator with true iff $p = \sigma_i$ and false otherwise. With the resulting formula, a monitor is instantiated and updated by calling the `eval` function (Algorithm 1) for every future event (τ_j, σ_j) until the instance has been decided as defined below. A formula object of the form $\varphi_1\ \mathcal{U}_t\ \varphi_2$ contains the attributes decided, $\overleftarrow{\varphi_1}$ (used to store $(\tau_k, [\![\varphi_1]\!](\rho, k))$ for all $i \leq k \leq |\rho|$), and $\overleftarrow{\varphi_2}$. We call a formula *decided* (decided

= true) if for any step j with $i \leq j \leq |\rho|$ and $\tau_j \leq \tau_i + t$ it is the case that either $\varphi_2 = \infty$ or $\varphi_1 = -\infty$. Then, we do not need to store any further instances in $\overleftarrow{\varphi_1}$ and $\overleftarrow{\varphi_2}$ because the truth value of φ only depends on the evaluations of φ_1 and φ_2 between step i and step j (Lemma 1). An upper bound on the space complexity of our algorithm is given by the number of subformulae we have to store and evaluate before φ is ultimately decided. Let $|\varphi|$ be the nesting depth of the temporal operators in the formula and t_l be the greatest deadline of all subformulae at nesting level l. In the worst case, the decision only occurs after all deadlines in φ have been surpassed in which case the complexity is $\mathcal{O}\left(\prod_{l=1}^{|\varphi|}(t_l \cdot \text{eventrate})\right)$, which is exponential in the nesting depth $|\varphi|$.

Algorithm 1: RVTL evaluation Algorithm

input : A formula object φ, the timestamp τ_i with respect to which φ is evaluated and the last received event (τ_j, σ_j).
output : The verdict $[\![\varphi]\!](\rho, 1)$ after observing (τ_j, σ_j)

Function `eval` $(\varphi, \tau_i, \tau_j, \sigma_j)$ **:**
  result $\leftarrow \infty$
  switch φ **do**
    case $b \in \mathbb{B}$ **do**
      if $b = \bot$ **then** result $\leftarrow -\infty$
    case $p \in \mathbb{P}$ **do**
      if $p \neq \sigma_j$ **then** result $\leftarrow -\infty$
    case $\varphi_1 \rightarrow \varphi_2$ **do**
      result $\leftarrow -1 \cdot (-1 \cdot$ `eval` $(\varphi_1, \tau_i, \tau_j, \sigma_j) \sqcup$ `eval` $(\varphi_2, \tau_i, \tau_j, \sigma_j))$
    case $\varphi_1 \, \mathcal{U}_t \, \varphi_2$ **do**
      `update` $(\varphi, \tau_j, \sigma_j)$
      if $\tau_j - \tau_i \leq t$ **and not** φ.decided **then**
        result $\leftarrow \tau_j - (\tau_i + t)$
      else
        result $\leftarrow \bigsqcup_{(\tau_k, v_k) \in \overleftarrow{\varphi_2}} \left(v_k \sqcap \bigsqcap_{\{(\tau_l, v_l) \in \overleftarrow{\varphi_1} | l < k\}} v_l \right)$
      end
    end
  end
return result

Lemma 3. *Let φ be a RVTL formula and $\rho \in (\mathbb{R} \times \Sigma)^*$ be a trace. Then*

$$\texttt{monitor}(\varphi, \rho) = \infty \iff \rho \in \text{good}(\varphi)$$
$$\texttt{monitor}(\varphi, \rho) = -\infty \iff \rho \in \text{bad}(\varphi)$$

In other words, the monitor decides the prefix problem.

Lemma 3 follows directly from Lemma 2 and the construction of Algorithm 1.

Algorithm 2: RVTL update Algorithm

Function `update` $(\varphi, \tau_j, \sigma_j)$ **:**
  $\overleftarrow{\varphi_1} \leftarrow \overleftarrow{\varphi_1} \cup (\tau_j,$`eval` $(\varphi_1, \tau_j, \tau_j, \sigma_j))$
  $\overleftarrow{\varphi_2} \leftarrow \overleftarrow{\varphi_2} \cup (\tau_j,$`eval` $(\varphi_2, \tau_j, \tau_j, \sigma_j))$
  foreach $(\tau_k, v) \in \overleftarrow{\varphi_1}$ **do**
    if $|v| \neq \infty$ **then** $v \leftarrow$ `eval` $(\varphi_1, \tau_k, \tau_j, \sigma_j)$
    if $v = -\infty$ **then** φ.decided $\leftarrow$ true
  end
  foreach $(\tau_k, v) \in \overleftarrow{\varphi_2}$ **do**
    if $|v| \neq \infty$ **then** $v \leftarrow$ `eval` $(\varphi_2, \tau_k, \tau_j, \sigma_j)$
    if $v = \infty$ **then** φ.decided $\leftarrow$ true
  end
return

V. Evaluation

The duration requirements for the various processes in a CATR fleet described in Section III can directly be translated into RVTL formulae as given in Table I.

φ_1	`orderCreated` $\rightarrow \Diamond_{t_1}$ `orderCompleted`
φ_2	`loaded` $\rightarrow \neg$`offline` $\mathcal{U}_{t_2}$ `unloaded`
φ_3	`docked` $\rightarrow \neg$`offline` $\mathcal{U}_{t_3}$ `undocked`
φ_4	`startDocking` $\rightarrow \Diamond_{t_4}$ `docked`
φ_5	`localized` $\rightarrow \Box_{t_5} \neg$`startLocalization`

TABLE I: Formal specifications

The problem setup favors a *violation monitoring* approach, where the monitor checks the whole trace for time steps i with $[\![\varphi]\!](\rho, i) = -\infty$. The atomic propositions are abstractions that must hold for all instances of the respective entity (order for φ_1, robot for φ_{2-5}) at runtime.

Figure 4 shows the result of evaluating the currently undecided instance of φ_2 for every robot over the first 200K steps in the dataset. The policy φ_2 states that it takes a robots a certain amount of time to deliver a load to its destination and during transport, the robot is not reported as being offline. Based on the explorative analysis performed in Section III, we set the deadline t_2 in φ_2 to 1000 and record time steps where an active instance is violated.

The results hint at the benefit of using RVTL for CATR monitoring. Over the first 80K steps (roughly 24 hours) only a single violation is recorded due to robot 7 going offline during transport. Then within the next 50K time steps we see multiple robots taking longer than usual for the completion of their transport jobs, and between steps 100k and 120k multiple robots are switched off due to a human operator resetting the fleet. From this observation, we draw two conclusions: (a) It is possible that a single fault source in the fleet's environment caused all delays by preventing the robots from reaching their goal locations in time. (b) The $\overline{\mathbb{R}}$-valued semantics provide more per-step information about the state of the whole system than two-valued approaches to monitoring.

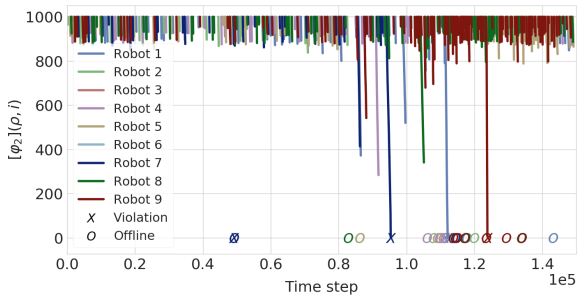

Fig. 4: $\overline{\mathbb{R}}$-valued evaluations of $[\![\varphi_2]\!](\rho_i, i)$ per monitor per time step for the first 150k events. Crossses mark violations of the specification, whereas circles indicate the corresponding robot being reported as offline.

Our proof-of-concept implementation is written in Python and has a per-step computation time around 0.5 ms for the presented trace and example specifications on a AMD FX-8350 @ 4.0GHz. Thus, even with commodity hardware we can increase the fleet size by orders of magnitude and still be able to monitor efficiently. Faster algorithms based on tableaus [21], timed automata [22] and rewriting techniques [20] have been discussed for real-time logics and adaptating the RVTL semantics to these is straightfoward. Furthermore, the monitoring can be easily scaled up with an increasing fleet size and number of formulas: With *parametric* monitoring [23], a single monitor is instantiated for every parameter binding in the formulas. If a new event is observed, only the relevant instances have to be updated, significantly reducing the per time step complexity.

VI. Conclusion

In this paper, we address the issue of monitoring a fleet of collaborative autonomous transport robots at runtime. Based on an analysis of logs from a real system we determined the boundary between normal and exceptional fleet behavior and concluded that we need the ability to express quantitative temporal relationshipts between system events. We propose RVTL, a fragment of the well-established logic MTL and develop a $\overline{\mathbb{R}}$-valued semantics which provides runtime information about the distance between the current time step and a temporal formula's deadline. RVTL's syntax is simple and the quantitative parameters correspond to intuitively applicable concepts such as deadlines and timeouts. As a proof of concept we implement a simple but trace-length independent algorithm for online monitoring of RVTL and evaluate it against the example trace. The results show that our approach is scalable with an increasing fleet size and the collection of individual $\overline{\mathbb{R}}$-valued verdicts presents a picture of the system health that contains early signs of runtime faults which are not visible from any individual verdict alone.

As a prospect for future work, it might be favorable to remove the dedicated monitoring appliance altogether and instead either use another consensus algorithm to determine a single monitoring robot at runtime or distribute the monitoring entirely. The latter brings new opportunities and challenges, e.g., with respect to how individual verdicts are consolidated to arrive at a collective monitoring verdict.

References

[1] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-time systems*, vol. 2, no. 4, 1990.

[2] M. Chai and B.-H. Schlingloff, "System monitoring with a five-valued ltl." *Journal of Multiple-Valued Logic & Soft Computing*, vol. 26, 2016.

[3] W. Ahrendt, J. M. Chimento, G. J. Pace, and G. Schneider, "Verifying data-and control-oriented properties combining static and runtime verification: theory and tools," *Formal Methods in System Design*, vol. 51, no. 1, pp. 200–265, 2017.

[4] N. Delgado, A. Q. Gates, and S. Roach, "A taxonomy and catalog of runtime software-fault monitoring tools," *IEEE Transactions on Software Engineering*, vol. 30, no. 12, pp. 859–872, Dec 2004.

[5] E. Bartocci, B. Bonakdarpour, and Y. Falcone, "First international competition on software for runtime verification," in *International Conference on Runtime Verification*. Springer, 2014, pp. 1–9.

[6] E. Bartocci, Y. Falcone, B. Bonakdarpour, C. Colombo, N. Decker, K. Havelund, Y. Joshi, F. Klaedtke, R. Milewicz, G. Reger *et al.*, "First international competition on runtime verification: rules, benchmarks, tools, and final results of crv 2014," *International Journal on Software Tools for Technology Transfer*, pp. 1–40, 2017.

[7] M. Delahaye, N. Kosmatov, and J. Signoles, "Common specification language for static and dynamic analysis of c programs," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM, 2013, pp. 1230–1235.

[8] M. d'Amorim and G. Roşu, "Efficient monitoring of ω-languages," in *International Conference on Computer Aided Verification*. Springer, 2005, pp. 364–378.

[9] M. Chai and B.-H. Schlingloff, "Monitoring with parametrized extended life sequence charts," *Fundamenta Informaticae*, vol. 153, no. 3, pp. 173–198, 2017.

[10] A. Kane, O. Chowdhury, A. Datta, and P. Koopman, "A case study on runtime monitoring of an autonomous research vehicle (arv) system," in *Runtime Verification*. Springer, 2015.

[11] D. D'Souza and P. Prabhakar, "On the expressiveness of mtl in the pointwise and continuous semantics," *International Journal on Software Tools for Technology Transfer*, vol. 9, no. 1, 2007.

[12] H.-M. Ho, "On the expressiveness of metric temporal logic over bounded timed words," in *International Workshop on Reachability Problems*. Springer, 2014.

[13] J. Ouaknine and J. Worrell, "Safety metric temporal logic is fully decidable," in *TACAS*, vol. 3920. Springer, 2006.

[14] ——, "Some recent results in metric temporal logic," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2008.

[15] H.-M. Ho, J. Ouaknine, and J. Worrell, "Online monitoring of metric temporal logic," in *International Conference on Runtime Verification*. Springer, 2014.

[16] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications," in *FATES/RV*. Springer, 2006.

[17] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE transactions on robotics*, vol. 25, no. 4, 2009.

[18] A. Goodloe and L. Pike, "Monitoring distributed real-time systems: a survey and future directions (nasa/cr-2010-216724)," 2010.

[19] J. Rushby, "Runtime certification," in *International Workshop on Runtime Verification*. Springer, 2008.

[20] P. Thati and G. Roşu, "Monitoring algorithms for metric temporal logic specifications," *Electronic Notes in Theoretical Computer Science*, vol. 113, 2005.

[21] T. A. Henzinger, "It's about time: Real-time logics reviewed," in *International Conference on Concurrency Theory*. Springer, 1998.

[22] J. Ouaknine and J. Worrell, "On the decidability and complexity of metric temporal logic over finite words," *arXiv preprint cs/0702120*, 2007.

[23] G. Rosu and F. Chen, "Semantics and algorithms for parametric monitoring," *arXiv preprint arXiv:1112.5761*, 2011.