Specification-guided Software Fault Localization for Autonomous Mobile Systems

Tomoya Yamaguchi, Bardh Hoxha, Danil Prokhorov
Toyota Research Institute-North America
Ann Arbor, U.S.
{tomoya.yamaguchi, bardh.hoxha, danil.prokhorov}@toyota.com

Jyotirmoy V. Deshmukh
University of Southern California
Los Angeles, U.S.
jyotirmoy.deshmukh@usc.edu

***Abstract*— Verification and validation are vital steps in the development process of autonomous systems such as mobile robots and self-driving vehicles, as they allow reasoning about system safety. In the domain of cyber-physical systems, techniques using formal requirements have been show to enable rigorous mathematical reasoning about system safety through techniques for automatic test generation and performance analysis. In this paper, we show that system-level and subsystem-level requirements can also enable fault localization in autonomous systems that use heterogeneous functional components. However, writing correct formal requirements is challenging and requires a significant investment of time, effort and most importantly, expertise. To address this issue, we propose a specification library for autonomous mobile systems called TLAM (Temporal Logic for Autonomous Mobility). Our contributions are twofold: We provide a library of parametric formal specifications at both the system-level and subsystem-level for typical subsystems in autonomous systems such as those for perception, planning and decision-making. The specification parameters encode the design trade-offs for such components. Second, we introduce a new fault localization technique based on these parametric specifications that identifies the likeliest subsystem that has a fault.**

***Index Terms*—Formal specifications, Formal verification, Autonomous systems, Verification & Validation, Fault Localization.**

I. Introduction

The software stack for autonomous mobile systems (AMSs) consists of many different components with distinct functionalities. Examples include subsystems for navigation, localization, perception, route and motion planning, and low-level vehicle control. For a given AMS, we can often specify its desired behavior using *mission specifications* and *safety specifications*. An example mission specification may express that the AMS should move from an initial region to a goal region within a specified amount of time. On the other hand, an example safety specification may express that the AMS should avoid collisions with static and dynamic obstacles in the environment. System-level specifications rarely delve into internal details of the AMS; however, system-level correctness of the AMS is contingent on each of the software subsystems satisfying their respective subsystem-level specifications. As the subsystems in an AMS are functionally diverse, their local specifications are also similarly diverse. An important research question is: *Can we express system-level and subsystem-level specifications in a uniform, mathematically precise and machine-checkable specification formalism?*

978-1-7281-9148-5/20/$31.00 © 2020 IEEE

An AMS can be considered as a Cyber-Physical System (CPS). Formal and semi-formal verification and validation (V&V) techniques for CPS applications have spawned significant amount of research in the past years [1]–[6]. A common thread in many of the research directions is the need for formal specifications; however, cultural practices in AMS development often preclude writing formal specifications at the system or subsystem levels. However, research has shown that creating specification libraries and an ecosystem of helpful tools can significantly accelerate industrial adoption of tools based on formal methods [7]–[9]. To address these two aspects, in this paper, we propose a specification library called Temporal Logic for Autonomous Mobility (TLAM), which is based on a popular requirement formalism known as Signal Temporal Logic (STL) [10]. STL is widely used for specifying CPS properties in various application domains [11]–[15]. See [16] for a survey. It already forms the basis of ST-Lib [7], a library of control-theoretic specifications for embedded control systems. The first contribution of this paper is to introduce a library of template specifications that developers can use to derive pertinent concrete specifications for their desired AMS application. The second contribution is to highlight the use of TLAM in addressing software fault localization. Our abstract idea for fault localization is extremely simple: if there is a violation of a system-level property, we try to identify if there is a subsystem-level specification that is violated. We do this by projecting the system-level behavioral trace onto subsystem-level (input/output) behavioral traces. However, several challenges arise. Typically, AMSs operate in uncertain, dynamic environments, and because they are closed-loop systems, the correctness of the AMS depends on whether design-time assumptions about the environment made by each subsystem are valid at run-time. A strict and inflexible subsystem-level specification that is oblivious of the operating environment would lead to excessive false positives due to environment assumptions being violated.

To address this challenge, we propose using not a single specification but a parameterized template representing a spectrum of specifications. For example, consider the STL specification $\mathbf{G}\,\mathbf{F}_{\,[0,d]}(x > 0)$, which says that it is **G**lobally

(i.e. at all times t) true that **F**inally within d seconds from t, at some time t', the value $x(t')$ exceeds 0. When the value of $d = 0$, then the specification reduces to checking if globally $x > 0$, and increasing the value of d progressively relaxes the specification. Thus, the template specification where d is a parameter represents an entire spectrum of concrete specifications. We can generalize this idea further and associate with any template specification a set of specifications that can be partially ordered using the logical implication operator. The main idea is to exploit this specification spectrum to perform fault localization. We now explain the key idea.

In our approach, we assume that the user provides a set of *correct system behaviors* by using TLAM. We then compute the generalized validity domain boundary of the specification spectrum, i.e., the set of parameter values for which the resulting concrete specification is marginally satisfied by the set of correct behaviors. We call this boundary the Requirement Spectrum Map (RSM). Now, given an actual system implementation, we can compare the RSM for each implemented subsystem with the baseline RSM. We then formulate a procedure to localize faults to a given subsystem by inspecting deviations of the implementation RSM from the baseline RSM. Finally, we enable fault localization not only on system architecture by using TLAM, but also at a behavioral level using the RSM. In order to demonstrate these ideas, we develop a detailed case study of a two-wheeled robot in a simulation setup using the Webots robotic simulator.

The layout of the paper is as follows. First, we introduce related works in Sec. II then show notation and definitions in Sec. III, then give a high-level overview of TLAM, the specification library for AMSs in Sec. IV. We discuss requirement-based fault localization with RSMs in Sec. V. In Sec. VI, we give our detailed case study, followed by conclusion (Sec.VII).

II. Related Work

Fault localization and debugging for cyber-physical systems has been an active area of research in software engineering. In [17] the authors present a survey of methods in this area. Statistical spectrum-based techniques [18], [19] have been used to detect faults by comparing the sets of passed and failed test cases. A common approach is to identify a similarity notion to find the closest distance between passed and failed tests and then use the difference for fault localization.

More recently, the fault localization problem has been studied for CPS models [20]–[22]. In [20], the authors utilize STL specifications to monitor and classify system behaviors developed in Simulink/Stateflow by associating time segments with variables that contribute to system failure. In [21], a similarity measure is presented for fault localization in CPS parameters and look-up maps. This similarity measure is then used to rank parameters with respect to system performance. In [22], [23], the authors provide a test generation process to localize faults and minimizing the set of test scenarios required for fault localization. These approaches try to localize bugs within a specific model using behavior traces and specifications, and hence are complementary to our approach. Our approach uses subsystem-level specifications to identify the most likely subsystem contributing to the error, and hence localizing faults within the subsystem is possible using statistical and static debugging approaches.

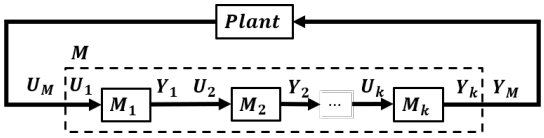


Fig. 1. An abstracted architecture of AMS.

In the robotics domain, it is typical to use traces consisting of low-level system information (such as sensor values, actuator signals, system variables etc.) [24]–[26] in order to analyze the reason for system failures. Typically, such information can be accessed by using a message-passing based operating system such as Robot Operating System (ROS) [27] that provides interfaces to log the communication between components. However, these are typically application-guided standard debugging practices that are not specification-guided. Here, we utilize a range of system requirements, ordered by difficulty of satisfaction, to generate a requirement spectrum map and use that to localize faulty components in an iterative development process.

III. Preliminaries

We now introduce required notation and definitions. A (discrete-time) signal is a time-stamped trace of the values taken by a variable. The formalization is below.

Definition 1 (Signal, Domains): Let $\mathbb{T} = (t_0, t_1, \ldots, t_N)$ be a finite subset of $\mathbb{R}^{\geq 0}$, where $\forall i$: $t_{i+1} > t_i$ known as the *time-domain*. A *signal* x is a function mapping $\mathbb{T}$ to some compact set X, known as the *value-domain*.

We next formalize the software architecture of an AMS.

Definition 2 (Autonomous mobile system architecture): An autonomous mobile system (AMS) M can be abstracted as a system with an input signal $\mathbf{u}_{\mathcal{M}}$ and an output signal $\mathbf{y}_{\mathcal{M}}$.The software architecture of $\mathcal{M}$ can often be described in terms of a tuple $(M_1, \ldots, M_k)$ (Fig 1), where: (1) each M_j is a software component encapsulating certain functional aspect, and (2) if $\mathbf{u}_j$ and $\mathbf{y}_j$ denote the input and output signals of the j^{th} component, then $\mathbf{u}_1 = \mathbf{u}_{\mathcal{M}}$, $\mathbf{y}_{\mathcal{M}} = \mathbf{y}_k$ and for all $j \in [1, k-1]$, $\mathbf{u}_j = \mathbf{y}_i$, for some $i < j$.

Essentially, the above definition models the software stack as a set of interacting subsystems with well-defined interfaces identifying the causal dependencies between subsystems. We assume that subsystems have an acyclic causal dependence captured by the last statement in the definition.

Example 1: For the detailed case study that we will present in Sec. VI, we consider a software stack that consists of five subsystems: a perception subsystem, a static path planner, a dynamic path planner, a decision-making system and a vehicle controller. The Perception system reads inputs from external sensors, and outputs distances to obstacles (static or dynamic) to the planners. The static planner takes as input the goal location and generates a route plan as a sequence of way-points. The dynamic planner operates while the AMS

is in motion, executing at each time step, and generating a plan based on any dynamic obstacles in the environment. At each time step, the decision-making system chooses the plan to execute based on the static and dynamic plans and feeds the reference velocities to the vehicle controller. The controller provides appropriate commands for actuating the vehicle motion to track the reference trajectory.

We now discuss system and subsystem-level requirements that specify desired behavior of the AMS. For the variables $\mathbf{u}_{\mathcal{M}}$, $\mathbf{y}_{\mathcal{M}}$, $\mathbf{u}_j$, and $\mathbf{y}_j$, let $U_{\mathcal{M}}$, $Y_{\mathcal{M}}$, U_j and Y_j denote their respective value domains. The set of all signals for a variable x with value domain X and time domain $\mathbb{T}$ is denoted by $\mathbb{T}^X$.

Definition 3 (Requirements): A *system-level* requirement $\varphi_{\mathcal{M}}$ is a non-empty subset of $\mathbb{T}^U_{\mathcal{M}} \times \mathbb{T}^Y_{\mathcal{M}}$. A *subsystem-level* requirement φ_{M_j} is a non-empty subset of $\mathbb{T}^{U_j} \times \mathbb{T}^{Y_j}$. We denote by $(\mathbf{u}_{\mathcal{M}}, \mathbf{y}_{\mathcal{M}}, t_0) \models \varphi_{\mathcal{M}}$ if the signals $\mathbf{u}_{\mathcal{M}}$ and $\mathbf{y}_{\mathcal{M}}$ starting at time t_0 satisfy $\varphi_{\mathcal{M}}$.

Signal Temporal Logic: We now give a high-level description of *Signal Temporal Logic (STL)* [10], [28] which we use as a basis for TLAM. The basic building block in STL is a *signal predicate* μ, which is a formula of the form $f(x) > c$, where x is a signal variable, and c is a real number. The syntax of an STL formula φ is defined recursively in (1), where I is an interval of the form $[a, b]$, where $a < b$ are real-numbers.

$$\varphi ::= \mu \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{F}_I \varphi \mid \mathbf{G}_I \varphi \mid \varphi \,\mathbf{U}_I\, \varphi \quad (1)$$

We now informally present STL semantics. The satisfaction of an STL formula over a signal is defined recursively in terms of the satisfaction of its subformulas. At a given time t, the signal predicate $f(x) > c$ is true iff $f(x(t)) > c$. Satisfaction of Boolean combinations of subformulas is defined in the obvious fashion. At time t, the formula $\mathbf{F}_{[a,b]}\varphi$ (resp. $\mathbf{G}_{[a,b]}$) holds if the formula φ is true for some time (resp. all times) $t' \in [t + a, t + b]$, while $\varphi_1 \,\mathbf{U}_{[a,b]} \varphi_2$ is true if φ_2 holds at some $t' \in [t + a, t + b]$ and for all $t'' \in [t, t')$, φ_1 is true.

Quantitative Semantics of STL: In addition to Boolean semantics, STL also has quantitative semantics that define a *robustness function* ρ that maps a formula φ, a time t and a signal x to the degree of satisfaction of φ over the suffix of x that starts at time t. If the robustness is non-negative (resp. negative), the formula is satisfied (resp. violated). While there are various quantitative semantics (see [16] for more details), we use the classical semantics defined in [29] for ease of exposition as presented below:

φ	$\rho(\varphi, x, t)$
$f(x) \geq c$	$f(x(t)) - c$
$\neg\varphi$	$-\rho(\varphi, x, t)$
$\varphi_1 \wedge \varphi_2$	$\min\left(\rho(\varphi_1, x, t), \rho(\varphi_2, x, t)\right)$
$\varphi_1 \vee \varphi_2$	$\max\left(\rho(\varphi_1, x, t), \rho(\varphi_2, x, t)\right)$
$\mathbf{F}_{[a,b]}\varphi$	$\sup_{t' \in [t+a, t+b]} \rho(\varphi, x, t')$
$\mathbf{G}_{[a,b]}\varphi$	$\inf_{t' \in [t+a, t+b]} \rho(\varphi, x, t')$
$\varphi_1 \,\mathbf{U}_{[a,b]} \varphi_2$	$\sup_{t' \in [t+a, t+b]} \min \left(\begin{array}{l} \rho(\varphi_2, x, t'), \\ \inf_{t'' \in [t, t')} \rho(\varphi_1, x, t'') \end{array} \right)$

IV. TLAM: Library of Requirements

We next present the abstract template-based STL specifications in Temporal Logic for Autonomous Mobility (TLAM) (Tab. I). The template requirements are formulated from our experience in developing autonomous mobile systems and by analyzing common patterns among requirements for different functional subsystems. We also discovered that some of the mission specifications that we use overlap with the comprehensive treatment on specifications for robotic missions in [30]. We assume that the set of all configurations or states of the AMS is described by a compact metric space X equipped with a distance metric d_X. We use $\mathbf{x}(t) \in X$ to denote the configuration of the AMS at time t. We assume that the distance d_X can be extended from (positive) distance to a notion of a *signed distance* $d^\star_X$ between the current configuration $\mathbf{x}(t)$ and some closed set S that is a subset of X, where the sign indicates if $\mathbf{x}(t)$ is in the interior or exterior of the set S.

Requirement for Flexibility in Specifications: In what follows, we present a number of TLAM specifications that represent a certain system quantity which is always bounded. The STL formula to indicate such a property is $\mathbf{G}\,\varphi$. However, the formula is often too strict, i.e., it would be impossible to find realistic systems that never violate such a property. In [31], the authors introduce the notion of robust LTL, which allows relaxation of invariance properties in LTL. We use a similar approach. There are two relaxations of $\mathbf{G}\,\varphi$: 1) *Permissible intermittent failure*, which is expressed as $\mathbf{G}\,\mathbf{F}_{[0,\tau]}\,\varphi$. This property requires φ to hold at least once in any interval of width τ, which is another way of requiring that φ be not false for more than τ time. 2) *Permissible initial failure*, which is expressed as $\mathbf{F}_{[0,\tau]}\mathbf{G}\,\varphi$. This property requires φ to always hold after at least some time τ elapses. Both properties coincide with $\mathbf{G}\,\varphi$ when $\tau = 0$. Furthermore, the formula φ may have signal predicates that could be relaxed (or tightened).

System-level Specifications.

Mission specifications: These specifications describe the objective of the task. Eq. (2) is a typical *reachability* specification[1] that requires the AMS to reach a final state in X_{goal} starting from an initial state in X_{init}. Given a prohibited region $B \in S$, Eq. (3) specifies that the AMS must avoid B.

Safety specifications:: These express physical safety of the AMS, including absence of collisions, bounds on critical system quantities (such as temperature, current drawn, velocity, etc.). Eq. (5) gives an example of an obstacle avoidance specification which avoids obstacle $O \in S$. Eq. (6) specifies that some signal $\mathbf{v}$ in the AMS always remains within some bounds $\mathbf{v}_{\min}$ and $\mathbf{v}_{\max}$ after a certain time τ.

[1]We remark that X_{init} and X_{goal} could be represented as a polyhedral region (conjunction of linear inequalities) or as an ellipsoidal region (sub level set of a suitable polynomial function), and in either case, the predicates $\mathbf{x} \in X_{init}$ and $\mathbf{x} \in X_{goal}$ can be represented as signal predicates in STL. We allow for different representations of the state sets by allowing an abstract interpretation of the membership operator $\in$.

TABLE I
TEMPORAL LOGIC FOR AUTONOMOUS MOBILITY (TLAM) LIBRARY.

Type	*Template*	**Formula**		Remark
Mission	$\mathsf{reach}(\mathbf{x}, X_{init}, X_{goal}, \in, \tau)$	$(\mathbf{x} \in X_{init}) \Rightarrow \mathbf{F}_{[0,\tau]}\,(\mathbf{x} \in X_{goal})$	(2)	The AMS reaches its goal location.
	$\mathsf{avoid}(\mathbf{x}, B, \in, \notin, \tau)$	$\mathbf{G}\,\big((\mathbf{x} \in B) \Rightarrow \mathbf{F}_{[0,\tau]}(\mathbf{x} \notin B)\big)$	(3)	The AMS avoids a region with some tolerance.
Safety	$\mathsf{no\_collision}(\mathbf{x}, O, d^{\star}_X, \tau, \varepsilon)$	$\mathbf{G}\,(d^{\star}_X(\mathbf{x}, O) > \varepsilon)$	(4)	The AMS strictly avoids a collision.
	$\mathsf{avoid\_collision}(\mathbf{x}, O, d^{\star}_X, \tau, \varepsilon)$	$\mathbf{G}\,\mathbf{F}_{[0,\tau]}\,(d^{\star}_X(\mathbf{x}, O) > \varepsilon)$	(5)	AMS is avoids a collision with some tolerance.
	$\mathsf{bounds}(\mathbf{v}, \mathbf{v}_{\min}, \mathbf{v}_{\max}, \tau)$	$\mathbf{F}_{[0,\tau]}\mathbf{G}\,(\mathbf{v}_{\min} < \mathbf{v} < \mathbf{v}_{\max})$	(6)	Eventually the AMS variable is always bounded.
Misc. Subsys.	$\mathsf{err\_obs}(\mathbf{v}, err, \tau, \varepsilon)$	$\mathbf{G}\,\mathbf{F}_{[0,\tau]}\,err(\mathbf{v}^*, \mathbf{v}) < \varepsilon$	(7)	Deviation from an observation is bounded in time.
	$\mathsf{err\_ref}(\mathbf{x}, err, \mathbf{x}_{\mathrm{ref}}, \tau, \varepsilon)$	$\mathbf{G}\,\mathbf{F}_{[0,\tau]}\,err(\mathbf{x}, \mathbf{x}_{\mathrm{ref}}) < \varepsilon$	(8)	Deviation from a reference is bounded in time.
	$\mathsf{no\_intersect}(\mathbf{p}, I, d^{\star}_X, \varepsilon)$	$d^{\star}_X(\mathbf{p}, I) > \varepsilon$	(9)	Trajectory does not intersection a geometric region.

Subsystem-level Specifications: An AMS may consist of several different kinds of subsystems such as perception from range or vision sensors, navigation subsystem for global route planning, map-based and landmark-based localization, high-level decision making (e.g. to decide which sub-task to pursue), trajectory/motion/path planning, and low-level actuation control. In TLAM we can specify behavioral properties for each subsystem, i.e. properties on the input/output behavior of the subsystem. For some subsystem such as range sensor/vision-based perception, TLAM can allow specifications in formalism such as TQTL (Timed Quality Temporal Logic) [32] or for certain controller subsystem allow formalism in Time Frequency Logic [33]. These and related logics can be viewed as extensions of STL. In this paper, we focus on STL-based subsystem specifications. We now present some of these selected *subsystem-agnostic* specifications.

Ground Truth-Based: These specifications characterize the tolerated distance between the actual system variable (i.e. the ground truth. Knowledge of the ground truth is a reasonable assumption in a simulator setup, or a carefully controlled physical experiment.) ($\mathbf{v}^*$) and the observed system variable ($\mathbf{v}$). Let err be a function used to compute the error between $\mathbf{v}^*$ and $\mathbf{v}$, then Eq. (7) requires that $err(\mathbf{v}^*, \mathbf{v})$ be bounded.

Reference Tracking: specifications measure the tolerated error between a given reference trajectory $\mathbf{x}_{\mathrm{ref}}(t)$ and the actual trajectory $\mathbf{x}(t)$ for a system. The form of these specifications Eq. (8) is the same as that of ground truth-based specs.

Static Geometric Relationships: These can also be expressed using STL. Suppose $\mathbf{p}$ is some point in the configuration space, and we wish to express that it lies outside of some closed set $I \in S$. Then, we can use Eq. (9) to express such a property. Note that we interpret $\mathbf{p}$ as a spatial signal $\mathbf{p}(t)$ that remains constant, so the STL formula without a temporal operator is interpreted at time 0, i.e. at $\mathbf{p}(0)$.

Now we can define at a high-level the idea of a machine-checkable requirement as a subset of possible input/output behaviors. We can then define what system-level M failures and subsystem-level M_k failures mean by using TLAM. In Sec. VI-A, we will give concrete instantiations of the specification templates. However, though these can deal with specification at the architecture level, the uncertainties of AMSs behavior remain a challenge, to be addressed by RSM in the next section.

V. REQUIREMENT SPECTRUM MAP

In this section, we describe how we can perform fault localization in AMSs using a spectrum of TLAM requirements defined by a Parametric Signal Temporal Logic (PSTL) to specify uncertain behaviors. PSTL is the logic obtained by replacing constants in an STL formula with parameters. A parameterized signal predicate of the form $f(x) \geq a$ contains a value parameter a, any temporal operator which can be associated with a parameterized time-interval $[\tau_1, \tau_2]$ with time parameters τ_1 and τ_2. A PSTL formula is denoted as $\varphi(\mathbf{p})$, where $\mathbf{p} = (p_1, \ldots, p_m) \in \mathcal{P}$ is the tuple of parameters appearing in the formula. A valuation function ν maps value parameters to the signal domain X, and the time parameters to $\mathbb{T}$ (ensuring $\tau_1 < \tau_2$). A PSTL formula $\varphi(\mathbf{p})$ along with a valuation ν for $\mathbf{p}$ defines an STL formula $\varphi(\nu(\mathbf{p}))$.

Thus, the PSTL spec $\varphi(\tau, a) = \mathbf{G}_{[0,\tau]}(x > a)$ has the parameters τ and a. The valuation function $\nu = \{\tau \mapsto 2, a \mapsto 20\}$ generates the STL formula $\varphi = \mathbf{G}_{[0,2]}(x > 20)$. The set of valuations is a partially ordered set with the order operator $\prec$, where $\nu \prec \nu'$ iff $\forall i : \nu(p_i) \leq \nu'(p_i)$. In what follows, we find the *monotonic fragment* of PSTL to be useful.

Definition 4 (Monotonic PSTL): A PSTL formula is called monotonically increasing if property Eq. (10) is true, and monotonically decreasing if Eq. (11) is true. If a formula is either monotonically increasing or decreasing, we call it a monotonic formula.

$$\forall x \in \mathbb{T}^X, \nu \prec \nu' \wedge x \models \varphi(\nu(\mathbf{p})) \Rightarrow x \models \varphi(\nu'(\mathbf{p})) \quad (10)$$

$$\forall x \in \mathbb{T}^X, \nu' \prec \nu \wedge x \models \varphi(\nu(\mathbf{p})) \Rightarrow x \models \varphi(\nu'(\mathbf{p})) \quad (11)$$

We introduce the notion of a Requirement Spectrum Map (RSM) which allows us to introduce flexibility in system- and subsystem-level specifications. As mentioned in Sec. IV, most of the specifications in TLAM have in-built flexibility through the use of parameters. We first motivate it with an example.

Example 2: Consider the spec. Eq. (7). When $\tau = 0$, then $\mathsf{err\_obs}(\mathbf{v}, err, 0, \varepsilon)$ is a strict requirement that does not permit occasional sensor noise. Nevertheless, in certain applications that can afford high-quality expensive sensors this may be

a valid specification. However, for less critical applications, it may be permissible to occasionally violate the strict error threshold by stating that in a time-window of size τ, there is at least one sensor measurement that satisfies the error threshold, i.e., $\mathsf{err\_obs}(\mathbf{v}, err, \tau, \varepsilon)$ is true.

In fact, in the above example, by allowing flexibility in the choice of the error threshold parameter ε and the time-window parameter τ, in addition to the syntactic structure of the STL formulas, we get a spectrum of requirements with varying (and incomparable) degrees of strictness. We can express the spectrum of requirements of interest to the user in the requirement spectrum map. We formally define the RSM.

Definition 5 (Requirement Spectrum Map): The RSM R is an $(n+1)$-dimensional map, where the $(i, j_1, \ldots, j_n)^{th}$ entry in the map is indexed by a PSTL formula φ_i that has at most n parameters and a valuation $\nu_{j_1,\ldots,j_n}$. For brevity, let $\mathbf{j}$ denote $(j_1, \ldots, j_n)$. The RSM satisfies the following properties:

$$\forall i, \forall \mathbf{j} : \varphi_i(\nu_{\mathbf{j}}(\mathbf{p})) \Rightarrow \varphi_{i+1}(\nu_{\mathbf{j}}(\mathbf{p})) \tag{12}$$

$$\forall i, \mathbf{j}, \mathbf{j}' : (\mathbf{j} \prec \mathbf{j}') \Rightarrow \big(\varphi_i(\nu_{\mathbf{j}}(\mathbf{p})) \Rightarrow \varphi_i(\nu'_{\mathbf{j}}(\mathbf{p}))\big) \tag{13}$$

We note that the RSM is closely related to the notion of the validity domain boundary for a PSTL formula [34]–[36]. For a given set of signals and PSTL formula $\varphi(\mathbf{p})$, its validity domain boundary represents the set of parameter valuations ν such that all signals satisfy $\varphi(\nu(\mathbf{p}))$, but an infinitesimal perturbation to ν (say ν') causes at least one signal to not satisfy $\varphi(\nu'(\mathbf{p}))$. While the validity domain is defined over the parameters of any PSTL formula, RSM is also defined over an implication lattice composed of a sequence of PSTL formulas. This lattice imposes a partial ordering with respect to strictness of the specifications. Thus, across the rows, we could use any sequence of implication-ordered formulas $\varphi_1 \Rightarrow \varphi_2 \ldots \Rightarrow \varphi_n$. For ease of exposition, in the rest of this section, we limit ourselves to RSMs where $i = 1$ and $n = 2$, i.e. there is a single PSTL specification which has two parameters.

Example 3: Consider the PSTL formula $\mathsf{err\_obs}(\mathbf{v}, err, \tau, \varepsilon)$. We can then pick $\nu(\tau) = 0$, d_1, or d_2 where $d_1 < d_2$, $\nu(\varepsilon) = e_1$ or e_2, where $e_1 < e_2$ as j_1^{th} and j_2^{th} indices of the RSM. One of the advantages of fixing the structure of the formulas is that the validity domain for a set of trajectories can be computed exactly and efficiently. We first define how a RSM R for an output signal x is computed and then extend on how this computation is done over a set of signals $\mathcal{X}$. Given a signal $x \in \mathcal{X}$, then the RSM R of x and consequently $\mathcal{X}$ as a function of parameters c and d is defined as follows:

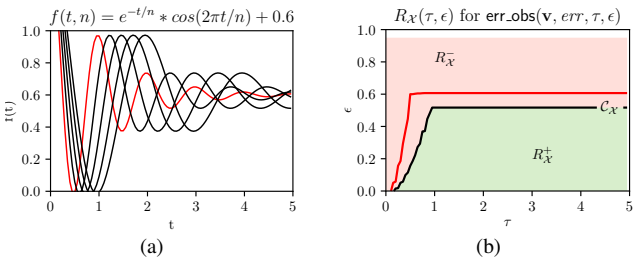
$$R_x(c,d) = \{\rho : \rho = \min_t(\max_{t'' \in [t,t']}(x(t'') - c) \text{ and } t' - t = d)\}$$

$$R_{\mathcal{X}}(c,d) = \{\rho : \rho = \min_{x \in \mathcal{X}} R_x(c,d)\} \tag{14}$$

Also, the set of satisfying and falsifying parameter valuations is defined as:

$$R^+_{\mathcal{X}} = \{(c,d) : \min_{x \in \mathcal{X}} R_x(c,d) > 0\}$$

$$R^-_{\mathcal{X}} = \{(c,d) : \min_{x \in \mathcal{X}} R_x(c,d) < 0\} \tag{15}$$



(a) (b)

Fig. 2. An example of RSM: (a) A simple transient response function where $n = 1$ (red) and $n \in \{1.25, 1.5, 1.75, 2\}$ (black). (b) The corresponding RSM with its 0-level boundary $\mathcal{C}_x$ (red) for a single trajectory when $n = 1$ and $C_{\mathcal{X}}$ with its 0-level boundary (black) for all trajectories where $n \in \{1, 1.25, 1.5, 1.75, 2\}$.

The RSM over system behaviors enables us to understand how well the system satisfies the spectrum of requirements in relation to its parameters. Here, a RSM $R^+_{\mathcal{X}}$ indicates the set of all satisfying parameter valuations and $R^-_{\mathcal{X}}$ indicates the set of all falsifying parameters. Of particular importance in an RSM is the 0-level set which separates the falsifying/satisfying parameter valuations. We refer to this set as the generalized validity domain boundary and define it as follows.

Definition 6 (Generalized Validity Domain Boundary $\mathcal{C}$): Given a set of output signals $\mathcal{X}$ and RSM R, the generalized validity domain boundary is defined as the 0-level set over all PSTL formulas φ_i and parameter valuations $\nu_{j_1,\ldots,j_n}$ such that

$$\mathcal{C}_{\mathcal{X}} = \{(i, j_1, \ldots, j_n) : R_{\mathcal{X}}(i, j_1, \ldots, j_n) = 0\} \tag{16}$$

The previous definitions enable us to define an RSM R as a tuple $(R^-_{\mathcal{X}}, \mathcal{C}_{\mathcal{X}}, R^+_{\mathcal{X}})$. To simplify the presentation, we may drop the subscript $\mathcal{X}$ and use only $\mathcal{C}$ to represent the Generalized Validity Domain Boundary for a set of trajectories.

Example 4: We illustrate the notion of a generalized validity domain boundary $\mathcal{C}$ through a simple example. Consider the function $f(t) = e^{-t} * cos(2\pi t) + 0.6$. In Fig. 2(b), we plot the function with respect to time and present its corresponding $\mathcal{C}$. For any parameter valuation in the red (resp. green) area, the signal falsifies (resp. satisfies) the specification. The $\mathcal{C}$, presented with the black line in Fig. 2(b), is the boundary between $R^+_{\mathcal{X}}$ and $R^-_{\mathcal{X}}$ for all trajectories and defines how robust the system is with respect to the robustness spectrum map.

We note that while in Ex. 3 we provide the exact solution for the RSM, for any other PSTL implication lattices, algorithms such as multi-valued binary search may be employed to approximate the RSM to find the generalized validity domain boundary in Eq. (16). We now describe how a RSM may be utilized for fault localization in iterative development.

A. *Fault Localization with* TLAM *and* RSM *in Development*

In a typical development process for such a complex AMS, a system and its subsystems are developed incrementally. The end product should satisfy a set of functional requirements for the system. Typically, this development process starts with a simple collection of subsystems that gradually are refined to compose a high-fidelity system. In the following, we show that our notion of a generalized validity domain boundary may be utilized for fault localization when iterating in system

development. First, we define a partially ordered set ($\mathcal{K}$, $\trianglerighteq$) over a generalized validity domain boundary $\mathcal{C}$. The binary operator $\trianglerighteq$ defines a reflexive, anti-symmetric and transitive relation over elements of $\mathcal{K}$.

Definition 7 (Ordering of Generalized Validity Domain Boundaries): Given two generalized validity domain boundaries $\mathcal{C}^i, \mathcal{C}^j \in \mathcal{K}$ and their corresponding RSM's R^i and R^j,

$$\mathcal{C}^i \trianglerighteq \mathcal{C}^j \text{ iff } R^{j-} \subseteq R^{i-} \tag{17}$$

The binary operator enables the comparison of $\mathcal{C}$'s at various stages of development. For a subsystem over 1 through n stages of development, a requirement-preserving development process maintains a strict ordering of $\mathcal{C}$'s where $\mathcal{C}^1 \trianglerighteq \mathcal{C}^2 \trianglerighteq ... \trianglerighteq \mathcal{C}^n$. In other words, the incremental development did not cause the set of parameter valuations for which the system is falsified to increase. If this ordering is violated, then the resulting RSM and the system output signals that have expanded the set of falsifying parameter valuations need to be analyzed as a suspicious behavior.

Now we can combine TLAM and RSM for fault-localization in an incremental development as follows.

1) Architecture perspective: TLAMs determine the false case from the architecture perspective while defining both system specifications $\varphi_{\mathcal{M}}$ and subsystem specifications φ_{M_k}.
2) Behavior perspective: RSMs specify uncertain behavior in each system and subsystem in more detail with respect to generalized validity domain boundary $\mathcal{C}$, which is calibrated by satisfied data, obtained from the incremental development.

In Sec. VI-B, we will give concrete instantiations of the fault-localization by using RSMs. In the next section we provide a case-study with a simple AMS.

VI. Case Study: Differential Wheeled Robot

AMSs have been studied extensively in the past, e.g., DARPA ground challenge [37] and urban vehicle challenge [38]. In this work, we utilize a simplified version of the system configuration presented in [39], which is a car-like non-holonomic vehicle with a Differential Wheeled Robot (DWR). We implement the DWR in a simulator setup in the Webots™ [40] robot simulator. The target robot used is *e-puck* [41] - a small robot 70 mm in diameter.

The kinematic model of the system is presented as follows.

$$\begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} = \begin{bmatrix} r_r/2 & r_l/2 \\ r_r/l & -r_l/l \end{bmatrix} \begin{bmatrix} \omega_r(t) \\ \omega_l(t) \end{bmatrix} \tag{18}$$

The wheel rotation right is $\omega_r(t)$, left $\omega_l(t)$, the diameter of right wheel is r_r, left r_l, and the tread of the two wheel is l. In addition, the position should be integral of $v(t)$, $\omega(t)$.

$$\begin{cases} x(t) = \int_0^t \cos\theta(\tau)d\tau + x(0) \\ y(t) = \int_0^t \sin\theta(\tau)d\tau + y(0) \\ \theta(t) = \int_0^t \omega(\tau)d\tau + \omega(0) \end{cases} \tag{19}$$

While the configuration space of the DWR is 3-dimensional, the control space is 2-dimensional, and the system is under-actuated. The architecture of the target DWR is pointed out in Fig. 3(a). The mission of the DWR is to reach a given goal, while avoiding collisions with obstacles and other agents in the work-space. Here is a brief description of each subsystem.

1) *Perception*: In an actual AMS, such as a range sensor, radar, and vision-based perception are commonly used with sensor fusion techniques such as Kalman filters or Particle filters for detection and localization [42]. For simplicity, we instead assume a perception that obtains the ground truth of the robot's position from the simulator, then adds some sensor noise to this position.
2) *Planner*: The planner subsystem switches between two different types of path planners: first, a way-points follower that attempts to track the path defined by a series of way-points; second, an obstacle avoider based on a Dynamic Window Approach (DWA) [43] (Fig. 3(c)). Finally a decision maker chooses a motion path from the candidate paths suggested by the two planners, then outputs a reference velocity to be used by the next actuation controller. Generally the ASMs has a global route planner, which generates way-points as a route. However, this experiment assumes a set of the way-points is given for ease of exposition.
3) *Vehicle controller*: A typical PID controller is implemented to control the motion actuators (motors) of the vehicle.

Definition 8 (Spatial Signals): The configuration or spatial state of the DWR is defined as $\mathbf{x} = (x, y, \theta)$, where $(x, y) \in \mathbb{R}^2$ and $\theta \in [0, 2\pi)$ are respectively the X and Y co-ordinates of the robot, and θ is its pose or orientation. We use $\ell(\mathbf{x})$ to denote the X-Y location of the DWR, i.e., the tuple (x, y). The *odometry spatial signal* of the DWR then is a sequence in $\mathbb{T}^X$, i.e., a sequence of time-value pairs. The *velocity spatial signal* of the DWR, denoted by $\dot{\mathbf{x}}$, is similarly a sequence which in each time t, $\dot{\mathbf{x}}(t) = (\dot{x}(t), \dot{y}(t), \dot{\theta}(t))$, i.e., the linear velocities in the X and Y direction and the angular velocity respectively. Finally, given an agent $\mathbf{a}_i \in \mathcal{A}$, its odometry spatial signal is denoted by $\mathbf{x}_{\mathbf{a}_i}$.

Definition 9 (Signed Distance to a Closed Set): For a point $\mathbf{p} = (p_x, p_y) \in \mathbb{R}^2$, let $\|\mathbf{p}\|_2$ denote its norm $\|\mathbf{p}\|_2 = \sqrt{p_x^2 + p_y^2}$ in Euclidean space. Given a configuration $\mathbf{x}$ and a closed set $S \subseteq \mathbb{R}^2$, we use $\mathbf{dist}(\mathbf{x}, S)$ and $\mathbf{depth}(\mathbf{x}, S)$ to denote the distance of the location of the configuration $\mathbf{x}$ to the set S and its complement.

$$\mathbf{dist}(\mathbf{x}, S) \stackrel{\text{def}}{=} \inf_{\mathbf{p} \in S} \|\mathbf{p} - \ell(\mathbf{x})\|_2$$
$$\mathbf{depth}(\mathbf{x}, S) \stackrel{\text{def}}{=} \mathbf{dist}(\mathbf{x}, \mathbb{R}^2 \backslash S)$$

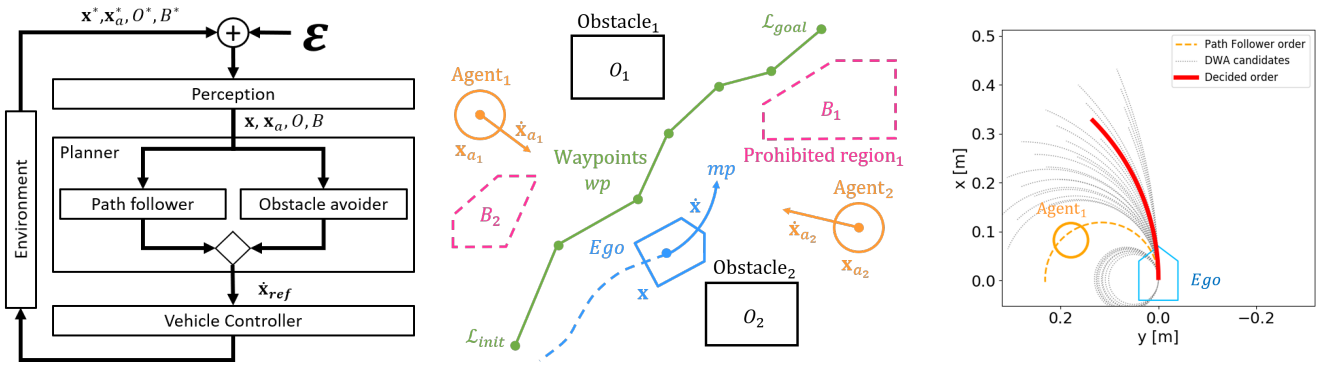
Using the above, $\mathbf{Dist}(\mathbf{x}, S)$ can be defined as follows:

$$\mathbf{Dist}(\mathbf{x}, S) \stackrel{\text{def}}{=} \begin{cases} -\mathbf{dist}(\mathbf{x}, S) & \text{if } \mathbf{x} \notin S \\ \mathbf{depth}(\mathbf{x}, S) & \text{if } \mathbf{x} \in S \end{cases} \tag{20}$$

A. Use of TLAM specifications for DWR

We now present TLAM in a specific DWR. The specifications are shown in Table II; the variable configuration of the DWR environment is visualized in Fig. 3(b).

Mission Specifications: System-level specifications are articulated using the ground truth X^* obtained from the simulator.



(a) The DWR architecture (b) The DWR variable map. (c) Dynamic Window Approach

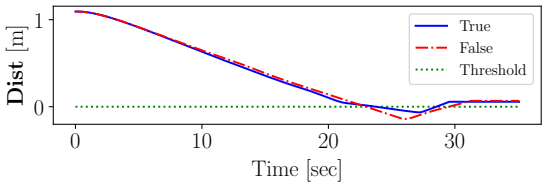
Fig. 3. The target differential wheeled robot system: (a) The system architecture which is based on Fig. 1. (b) The variable map of the system. (c) The behavior of Dynamic Window Approach.

TABLE II
SPECIFICATIONS FOR DWR.

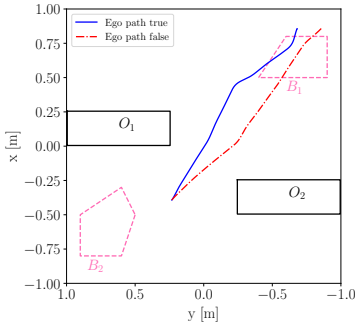
Level	Type	Specification	
System	Mission	$\mathsf{reach}(\mathbf{x}^*, \mathcal{L}_{init}{}^*, \mathcal{L}_{goal}{}^*,\ in^{\epsilon}\ , T_{\mathrm{dl}})$	(21)
System	Mission	$\bigwedge_{B_i{}^* \in \mathcal{B}^*} \mathsf{avoid}(\ell(\mathbf{x}^*), B_i{}^*,\ in^{\epsilon}\ ,\ not\_in^{\epsilon}\ , d)$	(22)
System	Safety	$\bigwedge_{O_i{}^* \in \mathcal{O}^*} \mathsf{no\_collision}(\ell(\mathbf{x}^*), O_i{}^*, \mathbf{Dist}, 0, 0)$	(23)
System	Safety	$\bigwedge_{\mathbf{a}_i{}^* \in \mathcal{A}^*} \mathsf{no\_collision}(\ell(\mathbf{x}^*), \ell(\mathbf{x}_{\mathbf{a}_i}{}^*), \mathbf{Dist}, 0, 0)$	(24)
Perception	GT-based	$\mathsf{err\_obs}(\ell(\mathbf{x}), err, d, \varepsilon)$	(25)
Perception	GT-based	$\bigwedge_{O_i \in \mathcal{O}} \mathsf{err\_obs}(O_i, err, d, \varepsilon)$	(26)
Perception	GT-based	$\bigwedge_{\mathbf{a}_i \in \mathcal{A}} \mathsf{err\_obs}(\ell(\mathbf{x}_{\mathbf{a}_i}), err, d, \varepsilon)$	(27)
Planner	Static Geom.	$\bigwedge_{\mathcal{L}_i \in \mathcal{W}, O_i \in \mathcal{O}} \mathsf{no\_intersect}(\mathcal{L}_i, O_i, \mathbf{Dist}, \varepsilon)$	(28)
Planner	Static Geom.	$\bigwedge_{\mathcal{L}_j \in \mathcal{W}, B_i \in \mathcal{B}} \mathsf{no\_intersect}(\mathcal{L}_j, B_i, \mathbf{Dist}, \varepsilon)$	(29)
Planner	Safety	$\bigwedge_{O_i \in \mathcal{O}} \mathsf{avoid\_collision}(mp, O_i, \mathbf{Dist}, d, \varepsilon)$	(30)
Planner	Safety	$\bigwedge_{B_i \in \mathcal{B}} \mathsf{avoid}(mp, B_i,\ in^{\varepsilon}\ ,\ not\_in^{\varepsilon}\ , d)$	(31)
Planner	Safety	$\bigwedge_{\mathbf{a}_i \in \mathcal{A}} \mathsf{avoid\_collision}(mp, \ell(\mathbf{x}_{\mathbf{a}_i}), \mathbf{Dist}, d, \varepsilon)$	(32)
Vehicle Controller	Ref. Tracking	$\mathsf{err\_ref}(\dot{\mathbf{x}}, err, \dot{\mathbf{x}}_{\mathrm{ref}}, d, \varepsilon)$	(33)

The DWR has two main mission specifications. In Eq. (21), we specify that starting from a location in $\mathcal{L}_{init}{}^*$, the DWR must reach a location in $\mathcal{L}_{goal}{}^*$ within T_{dl} time units. We define the *robust set containment* operator in^{ϵ} such that $\mathbf{x}\ in^{\epsilon}\ S$ if $\mathbf{Dist}(\ell(\mathbf{x}), S) > \epsilon$, i.e., if the signed distance of $\ell(\mathbf{x})$ is at least ϵ (i.e. the point should be at least ϵ inside the boundary of the closed set S). Analogously, we define the *robust exclusion* operator not_in^{ϵ} such that $\mathbf{x}\ not\_in^{\epsilon}\ S$ if $\mathbf{Dist}(\ell(\mathbf{x}), S) < -\epsilon$.

In Eq. (22), the DWR is also asked to avoid a set of prohibited regions specified through (ground-truth) coordinates $\mathcal{B}^* = \{B_1{}^*, \ldots, B_{|\mathcal{B}|}{}^*\}$. Here, the specification allows the DWR to trespass into the prohibited region for some small amount of time (d) before it is required to be robustly outside the prohibited region. This can be captured using robust set containment and exclusion operators in^{ϵ_1} and $not\_in^{\epsilon_2}$ as before. Thus, spec. Eq. (22) essentially says that if the DWR transgresses into any prohibited region $\mathcal{B}i$ by at most ϵ_1, within duration d, it must be at least ϵ_2 outside the prohibited region. This specification does not enforce the magnitude of the transgressions. If the DWR has an excursion into any B_i of greater than ϵ_1, the behavior is not specified. A second specification that bounds the degree of transgression can be added if required. The robustness of Eq. (22) for both weak and strong violations of the prohibited region, where $d = 1.5$ sec is shown in Fig. 4(a) and the map behavior is shown in Fig. 4(b). We find that the weak violation satisfies the spec. even though it violates the prohibited region for a



(a) The robustness of the mission spec. Eq. (22).



(b) The mission spec. Eq. (22) on map.

Fig. 4. System Mission results based on TLAM specifications for DWR: (a) System Mission results of Eq. (22) where $\mathbf{Dist}(\ell(\mathbf{x}^*), {B_1}^*)$. (b) Satisfying (blue) and falsifying (red) trajectories.

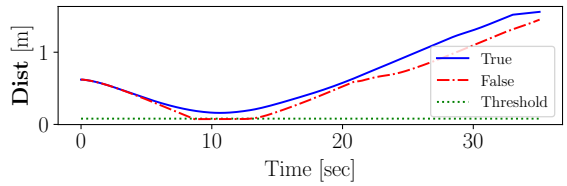


Fig. 5. System Safety result based on TLAM specifications DWR: System Safety results of satisfying (blue) and falsifying (red) trajectories with Eq. (24) where $\mathbf{Dist}(\ell(\mathbf{x}^*), \ell(\mathbf{x}_{\mathbf{a}_1}))$.

few seconds; in contrast, the strong violation does not satisfy.

Safety Specifications: The DWR safety specifications are specified in Eqs. (23) and (24). Those are also system level specs, thus we use the ground truth from the simulator. Both safety specifications express that the DWR should not have collisions, so we use the no_collision template spec. Eq. (4) from TLAM. Given ground truth values of static obstacles $\mathcal{O}^* = \{{O_1}^*, \ldots, {O_{|\mathcal{O}|}}^*\}$, in Eq. (23), we express that the DWR does not collide with any of them. We assume that the shape of each obstacle O_i is a closed set S, and we use **Dist** (as defined in Eq. (20)) as the signed distance required in the avoid_collision template. We use the values of $\tau = 0$ and $\varepsilon = 0$, which means that this specification strictly requires the system trajectory to not intersect with any of the obstacles. We assume that the environment of the DWR has a ground truth set of agents $\mathcal{A}^* = \{{\mathbf{a}_1}^*, \ldots, {\mathbf{a}_{|\mathcal{A}|}}^*\}$, where each agent defines its own odometry signal ${\mathbf{x}_{\mathbf{a}_i}}^*$. In Eq. (24), we invoke the no_collision template Eq. (4) to specify that no collisions with other agents are allowed. Consider the instance of Eq. (24). Here Fig. 5 plots distance between the ego and agent 1. The requirement is Eq. (24) with $\varepsilon = 0.08$ m.

Perception Specifications: The DWR perception specifications

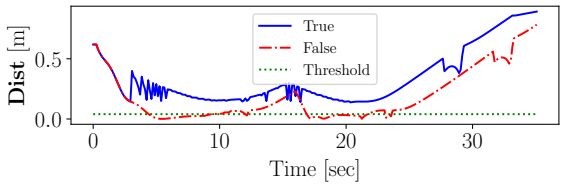


Fig. 6. Planner Safety result based on TLAM specifications DWR: Planner Safety results of satisfying (blue) and falsifying (red) trajectories with Eq. (32), where $\mathbf{Dist}(mp, \ell(\mathbf{x}_{\mathbf{a}_1}))$.

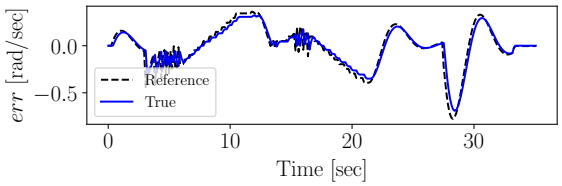
are specified in Eqs. (25)-(27). These perception specifications express that the error between any perception by the DWR and the ground truth from the simulator should be bounded; we use the err_obs template spec. Eq. (7) from TLAM for this purpose. We express in Eq. (25) that the DWR perceived odometry $\mathbf{x}$ is within the allowed error ε. We define the error specification in Eq. (26) for obstacles $\mathcal{O}$ and in Eq. (27) for other agents $\mathcal{A}$.

Planner Specification: The DWR planner specifications are specified in Eqs. (28)-(32). These are classified into static and dynamic motion planner specs based on the DWR architecture:

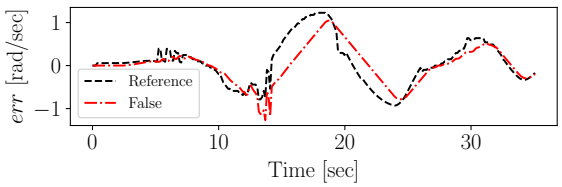
1) *Static planner*: the specs in Eqs. (28) and (29) express that the generated static path $\mathcal{W} = \{\mathcal{L}_1, \ldots, \mathcal{L}_{|\mathcal{W}|}\}$ does not intersect with obstacles $\mathcal{O}$ and prohibited regions $\mathcal{B}$. We use the no_intersect template spec. (9) from TLAM.
2) *Dynamic motion planner*: the dynamic motion planner finally decides the velocity of the DWR $\dot{\mathbf{x}}$. The expected path can be estimated from a linear extrapolation mp from velocities $\dot{\mathbf{x}}$ decided by the planner, i.e., $mp(\mathbf{x}, t_0, t_f) = \mathbf{x}(t_0) + (t_f - t_0)\dot{\mathbf{x}}(t_0)$. The motion trajectory that plans from current time $t_0 \in \mathbb{T}$ to some time $t_f \in \mathbb{T}^{>t_0}$ can be estimated with the prediction function. In other words, these motion planning specs express potential collision risk.
The DWR dynamic motion planner specs are given in Eqs. (30)-(32). We use avoid_collision template spec. Eq. (5) from TLAM. Collision with static obstacles $\mathcal{O}$ is in Eq. (30), with prohibited regions $\mathcal{B}$ is in Eq. (31), and with other agents $\mathcal{A}$ is in Eq. (32). Fig. 6 shows the result of a violating and non-violating case with $d = 5$ sec, $\varepsilon = 0.04$ m in the specs. We find the non-violating case maintains adequate distance from obstacles for all time, while the violating case transgresses the threshold ε often. Avoidance of prohibited regions $\mathcal{B}$ is specified in Eq. (31).

Vehicle Controller Specification: The DWR vehicle controller specification is given in Eq. (33). The specifications express a very typical controller issue in which the controller should closely follow the reference $\mathbf{x}_{\text{ref}}$. We use err_ref template spec. Eq. (8) from TLAM. Fig. 7 shows the results of true and false cases in specific $\dot{\theta}$, $\dot{\theta}_{ref}$ in $\mathbf{x}$, $\mathbf{x}_{\text{ref}}$ with $d = 2.0$ sec, $\varepsilon = 0.5$ rad/sec.

We have shown how TLAM can specify system and subsystem level specifications. We now show how RSM can support fault localization not only architecture level with TLAM but also uncertain behavior level in the next section.



(a) The true case of vehicle controller spec. Eq. (33).



(b) The false case of vehicle controller spec. Eq. (33).

Fig. 7. Vehicle controller Ref. Tracking result based on TLAM specifications DWR: (a) Controller tracking results of satisfying (blue) trajectory with Eq. (33) in specific angular velocity $\dot{\theta}$ where $err(\dot{\theta}_{ref}, \dot{\theta})$. (b) Falsifying (red) trajectory.

B. *Fault Localization* DWR *with* TLAM *and* RSM

The fault localization is a synergy of TLAM and RSM in this paper as we mentioned in Sec. V-A. In the previous section, Tab II shows system and subsystem level specifications with TLAM from the system architecture perspective. We pick up the representative specifications: Eq. (24) which evaluates no collision with agents as a system level specification, Eq. (27) which evaluates recognition errors of agent positions as a perception subsystem specification, Eq. (32) which evaluates whether the motion planned path has no collision with agents as a planner subsystem specification, and Eq. (33) which evaluates typical controller reference error as a vehicle controller subsystem specification. RSM is applicable to each subsystem specification; Eqs. (27), (32), (33) derive from them because those include flexible specification $\mathbf{G}\,\mathbf{F}_{[0,\tau]}\,\varphi$. Here, RSM preserves the generalized validity domain boundary $C_{\mathcal{X}}^{base}$ in Eq. (16), involves all stable cases from an incremental development based on Eq. (17). Finally, we can find TLAM based subsystem specification, which specifies the likeliest subsystem to have an issue while considering uncertainty of subsystem specifications by using RSM.

Here, a case of perception subsystem failure is shown in Fig. 8. An offset error in a few seconds is injected to the agent 1 position. That asserts a misclassification or an algorithmic issue and does not consider a sensor noise like a Gaussian noise. The time-sequence of recognition error is shown in Fig. 8(a), $err(\mathbf{x}^*, \mathbf{x})$ which is target of the spec. and Fig. 8(d) is its RSM. The parameters c and d come from RSM $R_x(c, d)$ in Eq. (14). The planner is specified by Eq. (32). The time-sequence of distance between planned path and the others is shown in Fig. 8(b) as $\mathbf{Dist}(mp, \ell(\mathbf{x}_{\mathbf{a}_i}))$ which is target of the spec. Fig. 8(e) is its RSM. The vehicle controller is specified by Eq. (33). The time-sequence of control reference error is shown in Fig. 8(c) where $err(\dot{\theta}_{ref}, \dot{\theta})$, $\dot{\theta} \subset \dot{\mathbf{x}}$, which is target of the spec. Fig. 8(f) is its RSM. Each generalized validity domain boundary $C_{\mathcal{X}}^{base}$ in RSM is defined based on system stable cases from an incremental development beforehand. Fig. 8(g) is plot of the result of system safety spec. Eq. (24). The case is injected into the perception component and finally the system level specification is violated in Fig. 8(g). Please see RSM in the perception subsystem Fig. 8(d). When the RSM of current trajectory $C_{\mathcal{X}}$ is satisfied previous condition, that must not exceed $C_{\mathcal{X}}^{base}$ and violate $R_{\mathcal{X}}^{+}$ area. However, only RSM of perception subsystem spec. is violated even though other subsystems' validity domain boundaries are satisfied. This false case shows that the perception has different behavior from boundary of stable cases and that the TLAM and RSM can identify that as the root cause.

A case of planner subsystem failure is shown in Fig. 9. In this case, the obstacle avoider (Fig. 3(a)) is intentionally disabled, thus only the way-points follower works. Here also the generalized validity domain boundary $C_{\mathcal{X}}^{base}$ in the RSM in the planner Fig. 9(e) is violated with RSM of the current trajectory $C_{\mathcal{X}}$ although the other specifications are satisfied.

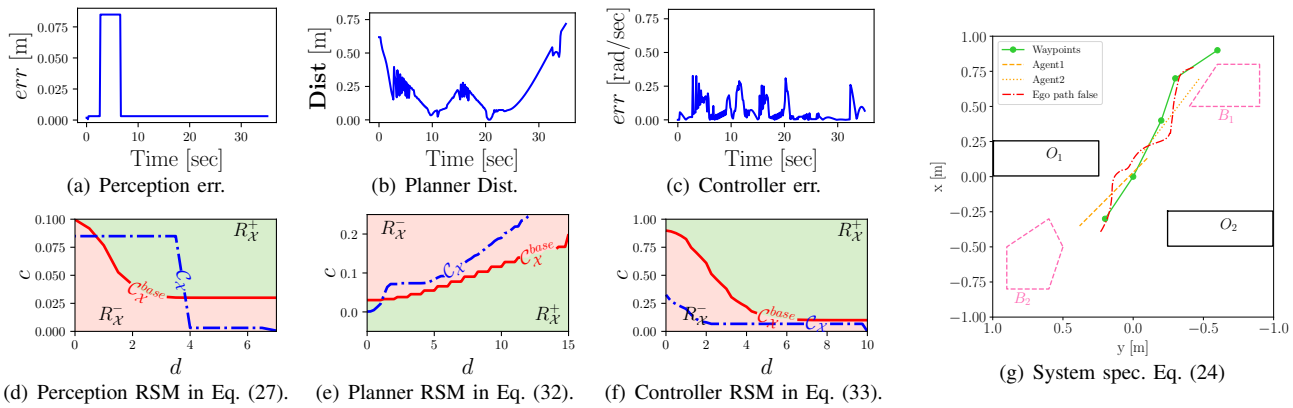
Also shown in Fig. 10 is a case of vehicle controller subsystem failure. Unreasonable PID gain is set as a fault injection. The generalized validity domain boundary $C_{\mathcal{X}}^{base}$ in the RSM in the vehicle controller Fig. 10(f) is violated with RSM of the current trajectory $C_{\mathcal{X}}$ because of the huge error between reference and control variables although the other specifications are satisfied.

Finally, no fault case is shown in Fig. 11. The system specification is satisfied while all generalized validity domain boundaries at the subsystem levels are satisfied. Thus, our proposed TLAM and RSM framework makes it possible to localize the root cause of system-level faults from both the architecture and behavior perspectives.

VII. Conclusion

We present TLAM and RSM for AMSs based on STL and show how this library can be used for fault localization. We give a detailed case study of a DWR that uses the library and show how we can identify faults in the system.

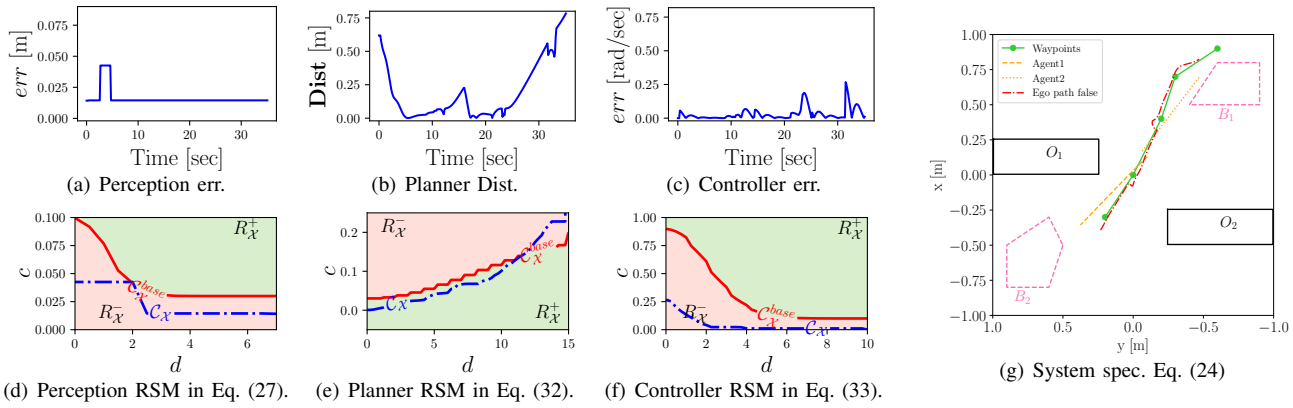
The automation and the overhead of computing the generalized validity domain boundaries could be improvement points. Several methods to improve computation efficiency of temporal logic [44], [45] can be referred. As further topic, multiple faulty in subsystems can be considered in contrast with this paper showed singular faulty in subsystem. Potentially, the identification of fault in such situation is challenging, because AMSs are closed-loop system and every faults effect to each other subsystem. A key challenge is to expand TLAM to include richer classes of perception requirements. Object recognition and computer vision tasks with deep neural networks (DNN) [46] are very common in this domain, and suitable specifications for such subsystems will be a vital portion of our future work. Also, real-world application in actual autonomous mobile system remains to be implementing. ROS is a well-known back-end on robotics domain. An open robot platform such as F1/10th [47] or HSR [48] would serve well for gaining reasonable practical, real-world experience.



(a) Perception err. (b) Planner Dist. (c) Controller err.

(d) Perception RSM in Eq. (27). (e) Planner RSM in Eq. (32). (f) Controller RSM in Eq. (33). (g) System spec. Eq. (24)

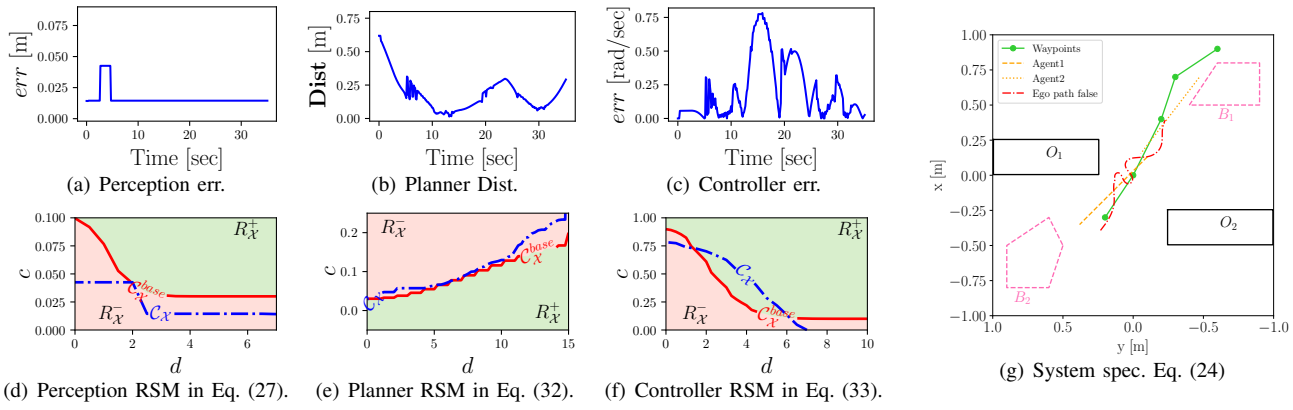
Fig. 8. A fault localization with TLAM and RSM in the perception failure case.



(a) Perception err. (b) Planner Dist. (c) Controller err.

(d) Perception RSM in Eq. (27). (e) Planner RSM in Eq. (32). (f) Controller RSM in Eq. (33). (g) System spec. Eq. (24)

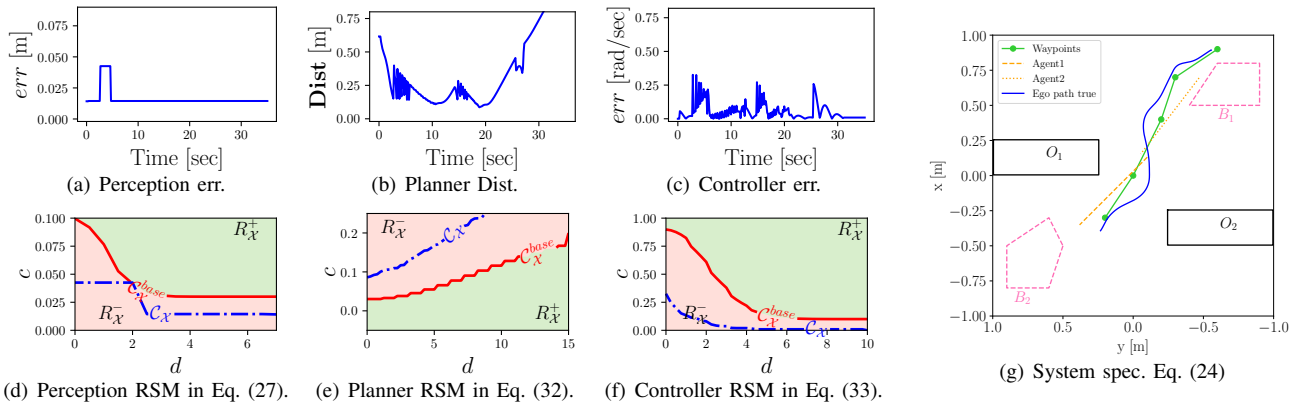
Fig. 9. A fault localization with TLAM and RSM in the planner failure case.



(a) Perception err. (b) Planner Dist. (c) Controller err.

(d) Perception RSM in Eq. (27). (e) Planner RSM in Eq. (32). (f) Controller RSM in Eq. (33). (g) System spec. Eq. (24)

Fig. 10. A fault localization with TLAM and RSM in the vehicle controller failure case.



(a) Perception err. (b) Planner Dist. (c) Controller err.

(d) Perception RSM in Eq. (27). (e) Planner RSM in Eq. (32). (f) Controller RSM in Eq. (33). (g) System spec. Eq. (24)

Fig. 11. A system green case of TLAM and RSM.

ACKNOWLEDGEMENT

The research performed at USC was sponsored in part by the National Science Foundation under the grants FMitF CCF-1837131, and CNS-1932620 and by Toyota Motor North America R&D.

REFERENCES

[1] R. Alur, "Formal verification of hybrid systems," in *Proceedings of the ninth ACM international conference on Embedded software*, 2011, pp. 273–278.

[2] ——, *Principles of cyber-physical systems*. MIT Press, 2015.

[3] P. Tabuada, *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media, 2009.

[4] J. V. Deshmukh and S. Sankaranarayanan, "Formal techniques for verification and testing of cyber-physical systems," in *Design Automation of Cyber-Physical Systems*. Springer, 2019, pp. 69–105.

[5] J. Kapinski, J. V. Deshmukh, X. Jin, H. Ito, and K. Butts, "Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques," *IEEE Control Systems Magazine*, vol. 36, no. 6, pp. 45–64, 2016.

[6] A. Platzer, *Logical analysis of hybrid systems: proving theorems for complex dynamics*. Springer Science & Business Media, 2010.

[7] J. Kapinski, X. Jin, J. Deshmukh, A. Donze, T. Yamaguchi, H. Ito, T. Kaga, S. Kobuna, and S. Seshia, "ST-Lib: A library for specifying and classifying model behaviors," SAE Technical Paper, Tech. Rep., 2016.

[8] T. Yamaguchi, T. Kaga, A. Donzé, and S. A. Seshia, "Combining requirement mining, software model checking and simulation-based verification for industrial automotive systems," in *2016 Formal Methods in Computer-Aided Design (FMCAD)*. IEEE, 2016, pp. 201–204.

[9] C. Eisner and D. Fisman, *A practical introduction to PSL*. Springer Science & Business Media, 2007.

[10] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 2004, pp. 152–166.

[11] Y. V. Pant, H. Abbas, R. A. Quaye, and R. Mangharam, "Fly-by-logic: control of multi-drone fleets with temporal logic objectives," in *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2018, Porto, Portugal, April 11-13, 2018*, 2018, pp. 186–197.

[12] D. Dey, D. Sadigh, and A. Kapoor, "Fast safe mission plans for autonomous vehicles," in *Proceedings of Robotics: Science and Systems Workshop*, 2016.

[13] H. Roehm, R. Gmehlich, T. Heinz, J. Oehlerking, and M. Woehrle, "Industrial examples of formal specifications for test case generation," in *Workshop on Applied veRification for Continuous and Hybrid Systems, ARCH@CPSWeek 2015*, 2015, pp. 80–88.

[14] G. Fainekos, B. Hoxha, and S. Sankaranarayanan, "Robustness of specifications and its applications to falsification, parameter mining, and runtime monitoring with S-TaLiRo," in *International Conference on Runtime Verification*. Springer, 2019, pp. 27–47.

[15] A. Dokhanchi, B. Hoxha, and G. Fainekos, "Formal requirement debugging for testing and verification of cyber-physical systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 2, pp. 1–26, 2017.

[16] E. Bartocci, J. V. Deshmukh, A. Donzé, G. E. Fainekos, O. Maler, D. Nickovic, and S. Sankaranarayanan, "Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications." Springer, 2017.

[17] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 2016.

[18] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proceedings of ASE*, 2005, pp. 273–282.

[19] W. E. Wong, V. Debroy, R. Gao, and Y. Li, "The dstar method for effective software fault localization," *IEEE Transactions on Reliability*, vol. 63, no. 1, pp. 290–308, 2013.

[20] E. Bartocci, T. Ferrère, N. Manjunath, and D. Ničković, "Localizing faults in Simulink/Stateflow models with stl," in *Proc. of HSCC*, 2018, pp. 197–206.

[21] J. Deshmukh, X. Jin, R. Majumdar, and V. Prabhu, "Parameter optimization in control software using statistical fault localization techniques," in *Proc. of ICCPS*, 2018, pp. 220–231.

[22] B. Liu, Lucia, S. Nejati, L. C. Briand, and T. Bruckmann, "Localizing multiple faults in simulink models," in *Proc. of SANER*. IEEE Computer Society, 2016, pp. 146–156.

[23] B. Liu, Lucia, S. Nejati, and L. C. Briand, "Improving fault localization for simulink models using search-based testing and prediction models," in *International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017, pp. 359–370.

[24] B. Annable, D. Budden, and A. Mendes, "Nubugger: A visual real-time robot debugging system," in *Robot Soccer World Cup*. Springer, 2013, pp. 544–551.

[25] P. Minnerup, D. Lenz, T. Kessler, and A. Knoll, "Debugging autonomous driving systems using serialized software components," *IFAC-PapersOnLine*, vol. 49, no. 15, pp. 44–49, 2016.

[26] M. De Rosa, J. Campbell, P. Pillai, S. Goldstein, P. Lee, and T. Mowry, "Distributed watchpoints: Debugging large multi-robot systems," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 3723–3729.

[27] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[28] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan, "Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications," in *Lectures on Runtime Verification*. Springer, 2018, pp. 135–175.

[29] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2010, pp. 92–106.

[30] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger, "Specification patterns for robotic missions," *IEEE Transactions on Software Engineering*, 2019.

[31] P. Tabuada and D. Neider, "Robust linear temporal logic," in *Proc. of CSL*, 2016, pp. 10:1–10:21.

[32] A. Balakrishnan, A. G. Puranic, X. Qin, A. Dokhanchi, J. V. Deshmukh, H. B. Amor, and G. Fainekos, "Specifying and evaluating quality metrics for vision-based perception systems," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1433–1438.

[33] A. Donzé, O. Maler, E. Bartocci, D. Nickovic, R. Grosu, and S. A. Smolka, "On temporal logic and signal processing," in *Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings*, 2012, pp. 92–106.

[34] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia, "Mining requirements from closed-loop control models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 11, pp. 1704–1717, 2015.

[35] E. S. Kim, M. Arcak, and S. A. Seshia, "Directed specifications and assumption mining for monotone dynamical systems," 2016, pp. 21–30.

[36] B. Hoxha, A. Dokhanchi, and G. Fainekos, "Mining parametric temporal logic properties in model-based design for cyber-physical systems," *STTT*, vol. 20, no. 1, pp. 79–93, 2018.

[37] M. Buehler, K. Iagnemma, and S. Singh, *The 2005 DARPA grand challenge: the great robot race*. Springer, 2007, vol. 36.

[38] ——, *The DARPA urban challenge: autonomous vehicles in city traffic*. springer, 2009, vol. 56.

[39] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[40] O. Michel, "Cyberbotics ltd. Webots™: professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004.

[41] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli, "The e-puck, a robot designed for education in engineering," in *Proceedings of the 9th conference on autonomous robot systems and competitions*, vol. 1, no. CONF. IPCB: Instituto Politécnico de Castelo Branco, 2009, pp. 59–65.

[42] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[43] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[44] A. Bakhirkin, N. Basset, O. Maler, and J.-I. R. Jarabo, "Paretolib: A python library for parameter synthesis," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2019, pp. 114–120.

[45] J. Cralley, O. Spantidi, B. Hoxha, and G. Fainekos, "Tltk: A toolbox for parallel robustness computation of temporal logic specifications," in *International Conference on Runtime Verification*. Springer, 2020, pp. 404–416.

[46] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. of NeurIPS*, 2012, pp. 1097–1105.

[47] M. O'Kelly, V. Sukhil, H. Abbas, J. Harkins, C. Kao, Y. V. Pant, R. Mangharam, D. Agarwal, M. Behl, P. Burgio *et al.*, "F1/10: An open-source autonomous cyber-physical platform," *arXiv preprint arXiv:1901.08567*, 2019.

[48] T. Yamamoto, K. Terada, A. Ochiai, F. Saito, Y. Asahara, and K. Murase, "Development of human support robot as the research platform of a domestic mobile manipulator," *ROBOMECH Journal*, vol. 6, no. 1, p. 4, 2019.