Checkmate: Fault Timing Localization for Multi-Robot Scenarios

Ippei Nishitani[1], Tomoya Yamaguchi[2], Bardh Hoxha[2]

***Abstract*— The development of control software for large-scale, complex systems remains a challenge. Control software development typically involves simulation or experimental testing to verify that system requirements are satisfied. However, if these tests detect failure data, that is, data that do not satisfy system requirements, the developer needs to perform system behavior analysis, fault localization and system modification. This process can be time-consuming and may require domain-specific knowledge. In this paper, we propose the Checkmate Timing Detection (CMTD) method to support the localization of unexpected behavior that causes failure. CMTD identifies anomalous behavior of the failure data by analyzing data obtained during the testing phase. The method can be utilized without specialized domain-specific knowledge. To demonstrate the approach, CMTD is applied to autonomous mobile robots operating in an indoor environment. We demonstrate through experimental evaluation that faulty system behaviors are accurately detected in a number of scenarios. The proposed method may also be utilized in control software applications and is expected to accelerate the software development process.**

I. INTRODUCTION

Robotic systems have seen tremendous advancements in recent years, and their control systems have become increasingly sophisticated. Going forward, robots are expected to work collaboratively in close proximity to humans in diverse mobility applications. The development of control systems for these systems involves a multi-phase process in order to guarantee that the system meets its intended functional specifications. This process begins with development of formal specifications, then moves on to the design and implementation of the software based on these specifications. Subsequently, the control system is tested through simulation and experimentation, following a typical v-model for development and integration. Whenever undesired behaviors are observed during the testing phase, the developer must investigate them through behavior analysis, fault localization and system modification, in order to rectify any issues that may arise. In robotics, these specifications dictate desired behaviors in both spatial and temporal contexts. For example, the specification may state that each robot should complete their *collect-and-deliver* task within n seconds while avoiding collision. Due to the complexity of behaviors encountered in real-world processes, the associated data may be extensive and varied, making manual fault detection time-consuming and requiring high expertise.

It is difficult to automatically detect unexpected behavior with limited domain-specific knowledge [1]. This is, in part, due to implicit behavior requirements that are not clearly defined beforehand. In Fig. 1, the three robots are expected to reach their respective goal locations. Aside from the imposed requirement that robots must reach their goals within a certain time period, there is an implicit behavior requirement that Robot 3 should not overtake Robot 2 under certain conditions. In order to detect such anomalies effectively, machine learning techniques can be applied to learn the implicit requirements and then use these learned models for anomaly detection.

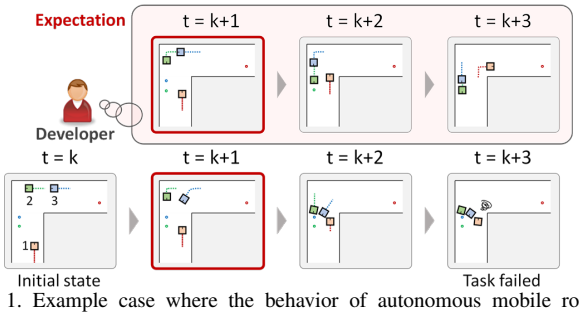


Fig. 1. Example case where the behavior of autonomous mobile robots passing each other deviates from the developer's expectation

Drawing on methods from systems verification and machine learning, this paper presents a fault localization technique that uses robustness-guided falsification and classification to detect anomalous behaviors in multi-robot systems. First, formal specifications written in Signal Temporal Logic [2] are used to collect and categorize robot behaviors and generate feature vectors. Then, similarity measures are employed to compare these vectors and detect unusual behaviors. A method for outlier detection and filtering is also utilized to improve performance. The results of this process can be used to enhance the safety, reliability, and performance of multi-robot systems.

Checkmate Timing Detection (CMTD) is an automated system developed to detect unexpected behavior in a set of multivariate signals, without requiring detailed domain-specific knowledge. This system goes beyond basic anomaly detection such as detecting spikes and flat lines and instead focuses on recognizing collective anomalies resulting from complex spatio-temporal interactions among multiple robots. Furthermore, the method can be used in an online fashion which enables anomaly detection and resolution while the systems are operating. For example, this approach can be used for well-established problems in robotics such as deadlock detection, to trigger a synchronization operation for the affected robots to resolve conflicts [3], [4], [5], [6].

The contributions of our study are listed as follows:

- We provide an automated method for identifying the causative signal and timing of unexpected behavior in requirement violating data without the need for detailed

[1]Toyota Motor Corporation, Japan; ippei_nishitani@mail.toyota.co.jp
[2]Toyota Motor North America, USA; {tomoya.yamaguchi,bardh.hoxha}@toyota.com

domain-specific knowledge.

- We propose an algorithm for filtering out outliers in success data in order to improve the accuracy of the CMTD fault localization technique. This method can be used as a stand-alone module to improve the precision of other fault localization techniques.
- We propose a relaxation of the anomaly detection method which is designed to improve scalability and, in certain cases, enables online anomaly monitoring, allowing for the implementation of resilient control architectures.

II. RELATED WORKS

Fault localization is an active area of research in a wide range of applications, from software and hardware debugging to robotics and distributed systems. Manual diagnosis is time consuming and expensive due to the complexity of data and interactions between components; therefore it is impractical.

This problem has been studied extensively in the software engineering domain, leading to a broad range of methods (e.g., see [1] for an overview). Approaches have ranged from combinatorial optimization techniques [7], to statistical [8], [9] and machine learning approaches [10], [11], [12], [13].

In this paper, we consider the problem from a robotics perspective, where beyond traditional software testing, there is a need to analyze over dynamic aspects of the problem. Specifically, faults that originate and can be classified as such by analyzing spatio-temporal behaviors of one or more robots. In this direction, various approaches have been proposed to address the problem. This includes approaches that take into consideration formal specifications of the system. In [14], the authors propose a debugging technique which combines testing, specification mining, and failure analysis to identify causes of failures in Simulink/Stateflow models. This technique can provide a time-ordered sequence of snapshots that show where the anomalous variables originated and how they propagated within the system. In [15], the authors utilize system-level and component-level specifications and specification mining to determine the most likely subsystem that causes failure. The method relies on a representative set of specifications to be defined and faults are detected only with respect to the specifications. In [16], a notion of shape expressions is proposed as a declarative language for specification and extraction of rich temporal patterns from time series data. This technique succeeded in detecting anomaly patterns in electrocardiograms and ringing in aircraft elevator control systems. However, its scalability is limited depending on the number and complexity of the signals in question. In [17], a method for finding temporal properties that lead to unexpected behaviors from labeled datasets is proposed. This technique does not require expert guidance and can be applied to data from simple traffic situations, like links and traffic lights. However, it is not clear if this method can be applied to data from more complex situations, such as an environment where multiple robots interact.

Different from the aforementioned methods, our proposed approach can be applied as a black-box for robotic applications with only a few parameters needing to be set, and limited domain knowledge required. Our method assumes faults to be imposed due to spatio-temporal interactions, and this allows us to simplify the process and enable analysis of numerous robots and complex robotic environments.

Deep learning approaches have also been utilized to solve the fault and anomaly detection problem. A survey of such methods based on deep neural networks is presented in [18], which categorizes anomalies into point, contextual and collective types. These approaches involve the use of Recurrent Neural Networks (RNNs) [19], [20], [21], Convolutional Neural Networks (CNNs) [22] and even attention models such as Transformers [23] for anomaly detection.

In general, Deep learning approaches require significantly more data than the method presented in this paper. Additionally, our method is specialized for multi-robotic systems, which not only detects anomalies but also identifies which robot interactions cause failures and when they occur.

III. PRELIMINARIES

A system Σ can be viewed as a function $\Delta_\Sigma : X_0 \times U^N \rightarrow Y^N \times \mathfrak{T}$ which takes as an input some initial condition, $x_o \in X_0$ and an input signal $u \in U$ and produces as output a signal $y : N \rightarrow Y$ and timing (or sampling) functions $\mathfrak{T} \subseteq \mathfrak{R}^+_N$. The produced signal is also called a discrete trajectory with a timing function $\tau : N \rightarrow \mathbb{R}_+$. The timing function τ is a monotonic, equidistant function, i.e., $\tau(i) < \tau(j)$ and $\tau(j) - \tau(i) = \tau(k) - \tau(j)$ for all i, j, k. The pair $\mu = (y, \tau)$ is referred to as a timed state sequence (tss) [24]. The set of all timed state sequences of a system Σ will be denoted by $L(\Sigma) = \{(y, \tau) \mid \exists x_o \in X_0 . \exists u \in U . (y, \tau) = \Delta_\Sigma(x_0, u)\}$.

Definition 1 (Partial Timed State Sequence): Given a timing interval $[t_0, t_1]$, a partial tss $\mu_{[t_0,t_1]} = (y(i), \tau(i))$ where for all $i, t_0 \leq \tau(i) \leq t_1$.

In this paper, we assume a constant sampling period $\Delta t > 0$ for the monitored system. For the fixed time period $\Delta t > 0$, for all $i \geq 0$, we have $\tau(i+1) - \tau(i) = \Delta t$ (or equivalently $\tau(i) = i\Delta t$). As a result, we can simply compute each time stamp $\tau(i)$ knowing the trace index (or simulation step) i by multiplication ($\tau(i) = i\Delta t$). Therefore, we use the trace index (simulation step i) as the reference of time.

IV. CHECKMATE TIMING DETECTION

This paper introduces Checkmate Timing Detection (CMTD), a method for detecting anomalous behavior in multidimensional time series data that may cause system requirement violations. We note that while there is no one-to-one relationship between anomalous behaviors and system requirement violations, there is often a high correlation between the two. The approach monitors the correlation between failure and success data over time, with the goal of identifying the point when recovery is no longer possible. We refer to this timing as Checkmate Timing (CMT).

CMTD is composed of four main components, as illustrated in Fig. 2. The first component (❶) of CMTD enables data collection through Search-Based Testing (SBT) [25], a well-known technique from formal methods. In SBT,

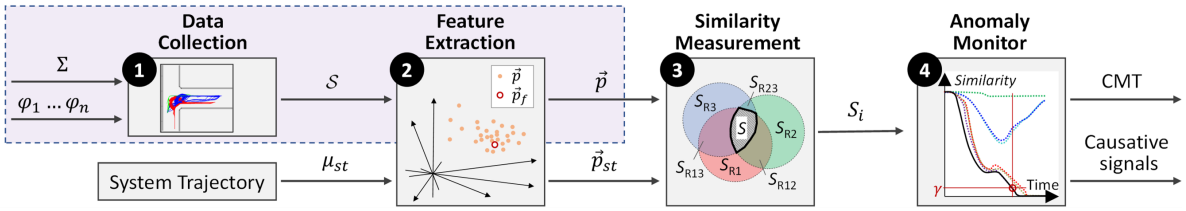


Fig. 2. Flowchart of CMTD. Given a system Σ and a set of specifications $\varphi_1 \dots \varphi_n$, a dataset S can be collected automatically in ❶ or provided by the user. Then, a feature extraction algorithm in ❷ is used for identifying and extracting important features in the data. We note that the first two steps can be preprocessed, as shown in the shaded area. Preprocessing splits up the trajectories in time window T_{ws} segments into representative features. This speeds up the anomaly detection algorithm and for small examples enables even computation as the system is running. The practitioner may select a system trajectory μ_{st} from $\mathcal{F}$ for investigation, or this can be generated in an online fashion while the system is running. Then, in ❸, a similarity measurement algorithm provides a measure that differentiate between the expected and unexpected features in the collected data. Finally, in ❹, if anomalous behavior is detected, the method returns CMT which is the timing when the anomaly is detected, and the causative signals or interactions.

formal requirements are used in a simulated environment where the system can be perturbed to generate different behaviors, each with its own robustness value with respect to the formal specifications. These values can be further exploited with stochastic optimization techniques to generate a diverse set of behaviors. Moreover, multiple requirements can be implemented to obtain both positive and negative behaviors for further investigation. The collected data is then processed in the second component (❷), in which characteristic features are extracted. Next, in component (❸), a similarity measure is proposed to differentiate between the expected and unexpected features in the collected data. Finally, a monitor is proposed in the fourth component (❹) for identifying scenarios of checkmate timing. Subsequently, each of these steps is described in further detail.

In this paper, we consider a running example of three robots navigating through an indoor T-junction environment (see Fig. 4). Each robot is located at a point opposite its destination, requiring them to traverse the environment in order to reach their goal. We consider the three robots and the environment as part of a larger system Σ and analyze how their individual control algorithms, which operate based on sensor input without explicit communication, affect the behavior of the full system. Our goal is to identify anomalous behaviors for improvement in the robots' performance when navigating through this type of environment.

A. Data Collection

Data collection (❶) can be done in an automated or traditional manner. Traditional methods involve unit testing of the system and manual classification of the results as passing or failing by a human operator. In contrast, an automated approach, as presented in this paper, entails providing a set of formal specifications and utilizing an automated process to determine the worst behaviors with regard to the given specs. We note that both approaches are compatible with CMTD.

A specification may be defined in a formal logic, such as Signal Temporal Logic (STL) [2]. This enables formalization of specifications such as "Go to region A within 30 seconds, while avoiding region B, and recharge within 10 minutes of battery charge going below 50%". Given a behavior of the system, represented as a timed state sequence μ, we can automatically check how robustly μ satisfies a collection of specifications $\phi_1 \dots \phi_n$ by using the theory of robustness of temporal logic specifications [25]. This enables a process called Search-Based Testing (SBT), which, by optimizing the behavior of the system over a set of input parameters, attempts to find violating behaviors. By doing so, the researcher can reduce the time taken to explore the reachable behavior space and increase the chance of finding a valid counterexample. This enables us to collect a set of behaviors $\Theta = \{\mu_1, \mu_2, \dots, \mu_m\} \subseteq L(\Sigma)$ and classify them as satisfying $\mathcal{S} = \{\mu \in \Theta | \mu \models \phi\}$ and falsifying $\mathcal{F} = \{\mu \in \Theta | \mu \not\models \phi\}$. This data is then used as a baseline for the anomaly detection algorithm. The success data $\mathcal{S}$ is used as reference data of CMTD, and the number of reference data is defined as M_{ref}.

B. Feature Extraction

From the collected data, a sliding window algorithm is applied to capture temporal properties. A feature extraction algorithm (❷) is then utilized for dimensionality reduction while preserving important characteristics. First, we utilize a sliding window technique for splitting the trajectory into smaller, overlapping windows. Let T_{ws} denote the window size and T_s denote the size between each window, then for a *tss* μ with size n, a sliding window method is used to generate partial timed state sequences $\Omega_{T_s}^{T_{ws}}(\mu) = (\mu_{[i,i+T_{ws}]})$ for all $i \in [1 : T_s : n - T_{ws}]$. Here $[1 : T_s : n - T_{ws}] = (1, 1 + T_s, \dots, 1 + H \times T_s)$ and $1 + H \times T_s + T_{ws} = n$. A feature extraction method is then applied to each element of $\Omega_{T_s}^{T_{ws}}(\mu)$. As a result, reference data vectors $\vec{p}$ of each element of $\Omega_{T_s}^{T_{ws}}(\mu)$ is obtained from $\mathcal{S}$. In our running example, for each robot, every T_{fe}-th sample for the (x, y) location of the robot is considered. Furthermore, the (x, y) location data is concatenated into one feature vector. As a result, for the motion of each robot, we have reference data vectors $\vec{p}$.

The window size should be set appropriately to capture anomaly behavior of the robots. To minimize the requirement for domain knowledge when selecting the optimal window size T_{ws}, we can employ a parameter mining algorithm (Alg. 1) to determine the best size for discriminating between satisfying and falsifying behaviors. This algorithm can be used to automate the process of finding the most suitable window size by exploring multiple possible parameter values and evaluating their performance. Fig. 3 shows the optimal window size for our running example. In our running example, the window size is automatically chosen based on the intuition that there is a difference in pause times between satisfying and falsifying behaviors.

Algorithm 1 OPTIMAL WINDOW SIZE

Input: Sets of trajectories $\mathcal{S}$, $\mathcal{F}$, Number of Robots N_s, speed threshold s_{min}, max window size max_{ws}
Output: T_{ws} ▷ optimal window size

function OPTWINDOWSIZE($\mathcal{S}, \mathcal{F}, N_s, s_{min}, max_{ws}$)
  $R \leftarrow 0$
  $T_{ws} \leftarrow 0$
  for $cand_{ws} = 1 : max_{ws}$ **do**
    $\mathcal{S}_R \leftarrow COUNT(Time_{th}(\mathcal{S}, N_s, s_{min}) \leq cand_{ws})/(|S|)$
    $\mathcal{F}_R \leftarrow COUNT(Time_{th}(\mathcal{F}, N_s, s_{min}) \leq cand_{ws})/(|F|)$
    if $R \leq \mathcal{S}_R/\mathcal{F}_R$ **then**
      $R \leftarrow \mathcal{S}_R/\mathcal{F}_R$
      $T_{ws} \leftarrow cand_{ws}$
    end if
  end for
  return T_{ws}
end function
function $Time_{th}(\mathcal{T}, N_s, s_{min})$
  for $\mu \in \mathcal{T}$ **do**
    for $n = 1 : N_s$ **do**
      $Time_{th}(idx(\mu), n) \leftarrow MAX\_TIME(\mu(n)[speed] \leq s_{min})$ ▷ finds longest period when robot is below a set speed threshold
    end for
  end for
end function

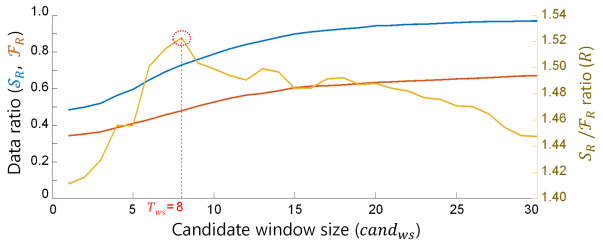


Fig. 3. The optimal window size for our running example (s_{min} = 0.01 m/s)

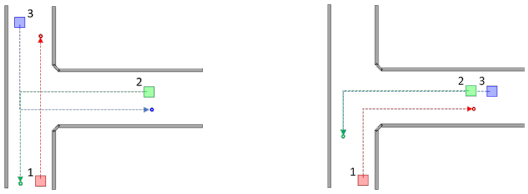
C. Similarity Measurement

For system trajectory μ_{st}, we repeat the process in steps ❶, ❷ to generate a target data vector $\vec{p}_{st}$. For the motion of each robot, similarity of $\vec{p}_{st}$ are computed by comparing with $\vec{p}$. Specifically, by counting $\vec{p}$ whose L2-norm with $\vec{p}_{st}$ is less than Euclidean distance threshold L_{th}, the number is used as similarity. Note that similarity for the motion of each robot indicates the number of similar data in $\vec{p}$.

For each interaction between robots, a similarity region (❸) is then constructed in each sliding window of the collected dataset. In the case that the practitioner desires to check all interactions between robots, then $2^{N^s} - 1$ different similarity regions are constructed. Specifically, $\mathfrak{S} = \{S_1, \ldots, S_{2^{N^s}-1}\}$. In our running example, for robots R1, R2, R3, we construct the following similarity regions $\mathfrak{S} = \{S_{R1}, S_{R2}, S_{R3}, S_{R12}, S_{R13}, S_{R23}, S_{R123}\}$ as shown in Fig. 2.

D. Anomaly Monitor

CMTD tracks the similarity of $\vec{p}_{st}$ for each similarity region and for each sliding window (❹). If at any point, the similarity measure of any of interactions is less than or equal to a threshold γ, we report the timing as CMT and report the corresponding interaction as anomalous. Lower values of a threshold γ result in earlier CMT and more sensitive results to anomaly behavior. In our running example, we are able to identify an agent or agents that cause system failure. Specifically, most of the failures are observed when the robots pass each other.



(a) (b)
Fig. 4. Test scenarios; (a) Experiment 1, (b) Experiment 2

E. Outlier Filtering

As we used CMTD in practice, we recognized that the success data $\mathcal{S}$ may include rare behaviors which are outliers. These rare behaviors sometimes lead to success, but often cause failure, thus reducing the performance of CMTD. As a solution to this problem, to improve the sensitivity of the anomaly monitor, we can use CMTD itself as a filter on the collected data to reduce/remove outliers. If checkmate timing is detected in an entry in the collected data, that entry is excluded from the reference data. This increases the sensitivity of CMTD when a new trajectory is processed.

F. Addressing Scalability Issues

We observe that as the number of robots increases, checking every interaction quickly becomes infeasible. Therefore, in those cases, we propose the use of Covering Arrays, a method used in combinatorial interaction testing which has been employed in the fields of software and hardware engineering, composite material design, and biological networks [7], [26]. They enable the reduction of the number of interactions needed to check, therefore the problem remains tractable even for a larger number of robots. For example, checking 2-way interactions for 10 robots requires only eight similarity regions. Checking 3-way interactions requires 18 tests [27]. We note that in software testing, case studies have shown that most failure cases (over 90%) are limited to a few interactions [28]. However, more case studies in robotics are required to generalize across domains. Using this approach, a lightweight, albeit incomplete, version of the method can be generated.

V. CASE STUDY

A. Overview of Experiments

We demonstrate CMTD on two scenarios where three robots pass each other at T-junction in an indoor environment. In the first scenario, shown in Fig. 4 (a), the initial conditions for the robots are defined such that each robot is in a separate corner of the environment. In the second scenario, shown in Fig. 4 (b), robots 2 and 3 are in close proximity and need to pass robot 1 to reach their goal. The robots are non-holonomic mobile robots equipped with a laser range sensor. The robots have a start-to-goal motion controller based on the A* algorithm. The robot control software is implemented in ROS, and the simulation is executed on Gazebo simulator. System requirements are set as formal specifications in STL.

TABLE I. Parameters of V & V Tests

V & V test terms		**Value**
Number of iterations of SBT		500 cycle
Scenario time (Specified goal time T_G)		120.0 s
Number of robots N_s		3
Robot	Maximum speed	0.3 m/s
	Maximum acceleration	0.3 m/s^2
	Laser range sensor position	Front center
	Sensor range	± 120 deg
Departure time of robot 1		7–37 s (Experiment 1, 2)
Departure time of robot 2		16 s (Experiment 1, 2)
Departure time of robot 3		7–37 s (Experiment 1) 16–26 s (Experiment 2)
Sampling step for robot positions data		0.1 s
Size of robot positions data n		1201 (120.0 s data)

TABLE II. Parameters of CMTD

CMTD terms	**Value**
Size of time window T_{ws}	80 (8.0 s window)
Sliding step between each window T_s	10 (1.0 s)
Feature parameter extraction step T_{fe}	10 (1.0 s)
Feature parameter dimension D	18
Similarity judgment distance threshold L_{th}	$0.3\times\sqrt{D}$
Similarity threshold for CMT γ	$0.01\times M_{ref}$

The system requirements, expressed in STL, ϕ_{ex1} and ϕ_{ex2} for experiments 1 and 2 are given as:

$$\phi_{ex1} : \bigwedge_{i=1}^{3} \mathbf{F}_{[0,T_G]}(xr_i \in GR_i) \tag{1}$$

$$\phi_{ex2} : \mathbf{F}_{[0,T_G]}(xr_1 \in GR_1) \wedge \bigvee_{i=2}^{3} \mathbf{F}_{[0,T_G]}(xr_i \in GR_i) \tag{2}$$

In experiment 1, all three robots must reach their respective goal regions within the specified goal time T_G; that is, for each robot denoted by i, the state xr_i must be within its corresponding goal region GR_i. In experiment 2, Robot 1 as well as either Robot 2 or 3 must reach their respective goals within the same time frame.

At this stage of the development process, a V&V technique is used to verify that the system satisfies system requirements. We utilize Search-Based Testing (SBT) [25], which uses stochastic optimization algorithms to find undesired behaviors of the system with respect to formal specifications. The search space, or the input to the SBT algorithm, is the departure time for robots 1 and 3. SBT was performed using the verification toolbox Breach [29] and the optimization tool HEEDS using the optimization algorithm Sherpa [30]. TABLE I shows the parameters of experimental setup.

In addition, in order to confirm the effectiveness of CMTD in various cases, we extract various failure data by using a clustering method as follows. We use Ward's method [31] to divide the failure data into 16 clusters. For each cluster, we extract the data point that is closest to the average of all data points in that cluster. In this way, we obtain 16 representative failure data points.

B. Experimental Parameters

The proposed CMTD algorithm has several parameters that need to be set, which the authors determine through trial and error with limited domain-specific knowledge. These parameters include those related to extracting features of the robot trajectories as well as those related to calculating the similarity between trajectories. See TABLE II for experimental parameters of CMTD.

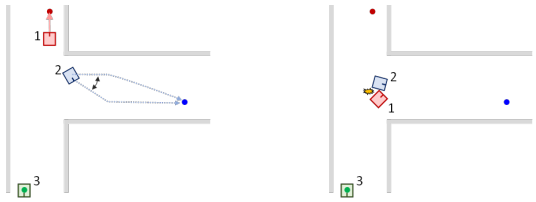


Fig. 5. Failure cases; (a) One robot stopped, (b) Two robots stopped

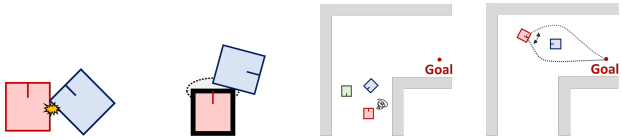


Fig. 6. Suspicious behavior patterns for true CMT setting; (a) Collision, (b) Virtual bumper, (c) Path planning failure, (d) Path chattering

To conduct an experimental evaluation of CMTD, we compare our method to the true value of CMT computed using detailed domain-specific knowledge. In the cases shown in Fig. 5 (a), one robot stopped due to frequent switching of paths, while in the case shown in Fig. 5 (b), two robots stopped due to a collision. We consider four types of behavior patterns as shown in Fig. 6.

- SB1 Collision: Two robots have collided because the shortest distance between them is less than the sum of their radii.
- SB2 Virtual bumper: When the robot detects an obstacle in front of it, the command speed is set to 0 through the virtual bumper flag.
- SB3 Path planning failure: The path planning state flag has detected that path planning has failed.
- SB4 Path chattering: The robot has stopped due to frequent switching paths, as indicated by the robot speed, the robot angular velocity, and the amount of change in the path switching.

The timing when SB1-SB4 occur is set as true CMT. True CMT can be caused by singular or multiple robots. When more than one robot exhibit suspicious behaviors (in this case SB1-SB4) within CMT to CMT+10 seconds, they are all are regarded as causative signals.

C. Experimental Design

In order to evaluate the effectiveness of the CMTD algorithm, we compare it with two additional methods: Method A, which is CMTD without the outlier filtering function; Method B, which is defined as follows: The system requirements specify that robots reach the goal within a specified time. If a robot does not reach the goal, it is considered that the robot stopped at a point other than the goal is the cause of the requirement violation. The timing when the speed finally falls below the threshold (0.05 m/s) at a point other than the goal is defined as CMT as shown in Fig. 7. In addition,

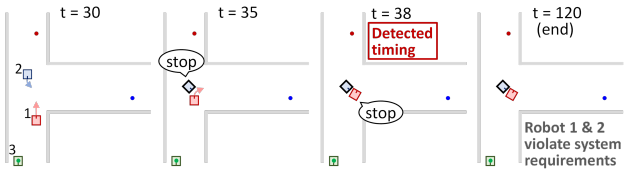
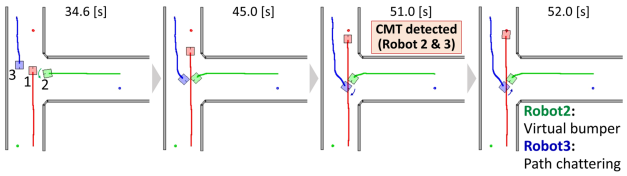
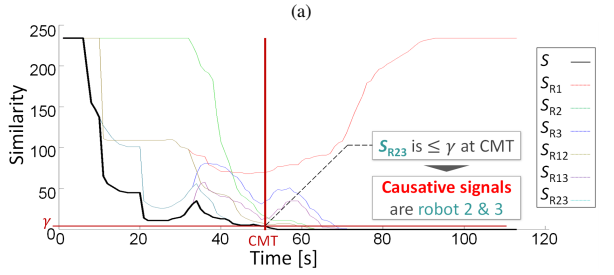


Fig. 7. CMT detection by method B



(a)



(b)

Fig. 8. CMT detection case by the proposed method; (a) Robots behavior, (b) Time series similarity data. Simulation video can be found at https://youtu.be/14ogkaKUhyY.

all robots whose speed is below the threshold from CMT to CMT+10 seconds are regarded as causative signals.

D. Evaluation Criterion

To verify the accuracy of CMT, the experimental results are shown in observed-predicted plots and histograms of the CMT prediction errors. We consider CMT as accurate if they are within 5 seconds of the true CMT and define the accuracy rate by ACC. The 5-second time bound is considered since it is less than the time window size $T_{ws} = 8s$ and sufficiently narrows down the problem for the practitioner. In addition, the results are evaluated using the mean absolute error (MAE), to consider the effects of outliers:

$$MAE = \frac{1}{n_t}\Sigma_{i=1}^{n_t}|t_i^T - t_i^Y| \tag{3}$$

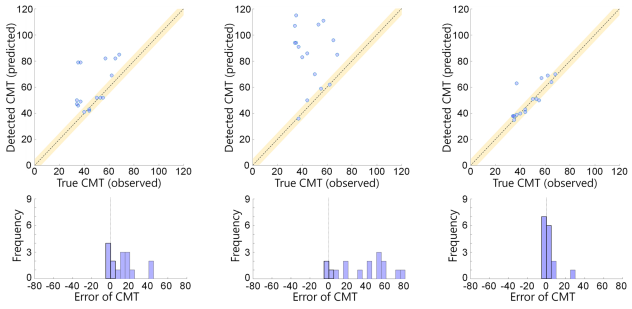
where n_t is the number of representative failure data points, t_i^T and t_i^Y are true CMT and detected CMT, respectively. For the causative signals, an F1 score of a multi-label classification algorithm considering partial correctness [32] is used for evaluation:

$$F_1 = \frac{1}{n_t}\Sigma_{i=1}^{n_t}\frac{2|l_i^T \cap l_i^Y|}{|l_i^T| + |l_i^Y|} \tag{4}$$

where l_i^T and l_i^Y are the true label and the detected label of the causative signal, respectively.

E. Experimental Results and Discussion

In experiment 1, we obtained 150 failure data and 350 success data. For the proposed method, 234 reference data (66.7%) were extracted from success data using the outlier filtering function. Fig. 8 highlights an example where CMT



(a) (b) (c)

Fig. 9. Observed-predicted plot and histogram of the error of CMT (experiment 1); (a) Method A, (b) Method B, (c) Proposed method

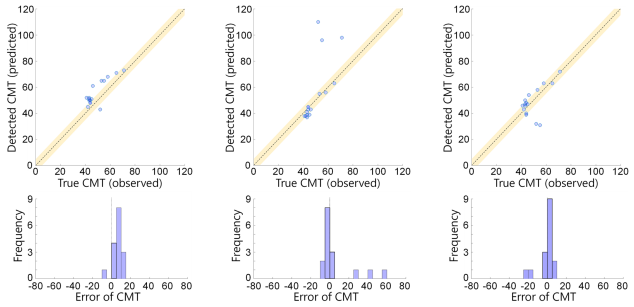
TABLE III. Analysis of Experiment 1

Methods	ACC of 16 data	MAE [s]	F1 score
Method A	37.5% (6/16)	12.5	0.81
Method B	18.8% (3/16)	37.3	**0.94**
Proposed method	**81.3% (13/16)**	**4.4**	0.88

is detected by the proposed method. There, robot 2 stopped due to its path changing at 34.6 s, but CMT was not detected because this behavior appeared in the reference data and does not lead to future failures. The proposed method was able to detect CMT when robots 2 and 3 stopped due to virtual bumper and path chattering behaviors. Therefore, the proposed method succeeded in fault detection by extracting two suspicious behaviors occurring at the same time as the true CMT.

In addition, we compared the experimental results of method A, method B, and the proposed method, as shown in Fig. 9 and TABLE III. Fig. 9 shows the Observed-predicted plot and histogram of the error of CMT. Bold face in TABLE III indicates the best performance. As shown in Fig. 9 and TABLE III, method A and method B failed to detect the true CMT in most of the data. As shown in Fig. 9 (a), method A detects most of the CMTs later than the true CMT. One possible reason for this discrepancy is that method A uses all success data as reference data, which means that even rare behaviors contained within that data are regarded as satisfying the behavior requirements. For the causative signal detection, method B has the highest F1 score, but it is difficult to extract suspicious behavior because the detected CMT is significantly different from the true CMT value. Furthermore, the proposed method obtained the best performance in terms of ACC and MAE, while its F1 score was also satisfactory for correctly identifying causative signals. Since the proposed method gives more accurate results than method A, it was confirmed that the outlier filtering function improves the identification accuracy of CMT. The results of experiment 1 show that the proposed method can accurately extract suspicious behaviors with CMT and appropriate causative signals by using success data, even without detailed domain-specific knowledge.

In experiment 2, we changed the robot passing scenario and obtained 36 failure data and 464 success data. For the proposed method, 440 reference data (94.8%) were extracted from success data using the outlier filtering function. We compared the experimental results of method A, method B



(a) (b) (c)

Fig. 10. Observed-predicted plot and histogram of the error of CMT (experiment 2); (a) Method A, (b) Method B, (c) Proposed method

TABLE IV. Analysis of Experiment 2

Methods	ACC of 16 data	MAE [s]	F1 score
Method A	25.0% (4/16)	6.5	0.85
Method B	68.8% (11/16)	10.2	0.84
Proposed method	**75.0% (12/16)**	**6.3**	**0.87**

and the proposed method, as given in Fig. 10 and TABLE IV. As shown in TABLE IV, the proposed method has the best score in *ACC*, MAE and F1 score compared to method A and method B.

Even in the case when CMT is not correctly determined by the proposed method, the results returned can be useful for exploring problematic behavior. For instance, in Fig. 10 (c) and Fig. 11, we show two outlier cases, where CMT was reported earlier than the true CMT, however, this particular configuration of the robots leads to a racing condition which will likely lead to failure in the future. With this insight, it is possible to develop resilient architectures that can adapt control parameters such as speed and turning rate when a racing condition is detected. In this way, the proposed method was able to provide new insights for fault localization that even approaches with detailed domain-specific knowledge may not suggest. Thus, the results of experiment 1 and 2 confirm that the proposed method can detect suspicious behaviors of the causative signals in various situations.

VI. CONCLUSION

The Checkmate Timing Detection method proposed in this study can accelerate the software development process by automatically extracting unexpected behavior of the causative signal from requirement-violating data collected during validation and verification. An outlier filtering function is introduced to improve the accuracy of identifying unexpected behavior by excluding data that includes rare behavior. As future work, we will explore the applicability of the proposed method to various operating environments with dozens of robots running simultaneously.

ACKNOWLEDGMENT

We are grateful to Yuji Date and Yuta Watanabe of TOYOTA MOTOR CORPORATION for supporting the construction of the simulation environment.

REFERENCES

[1] 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.

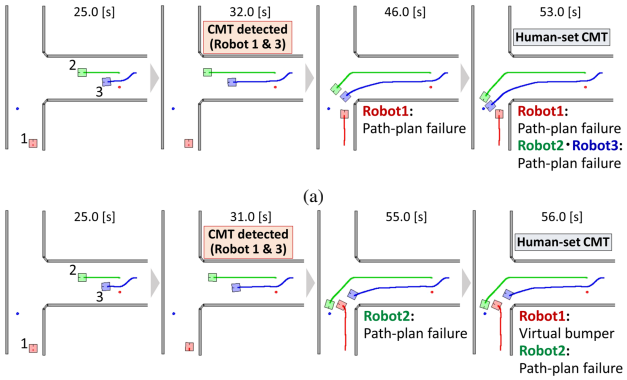


Fig. 11. Outlier cases of the proposed method; (a) Case 1, (b) Case 2

[2] 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.

[3] D. E. Soltero, S. L. Smith, and D. Rus, "Collision avoidance for persistent monitoring in multi-robot systems with intersecting trajectories," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 3645–3652.

[4] J. S. Grover, C. Liu, and K. Sycara, "Deadlock analysis and resolution for multi-robot systems," in *Algorithmic Foundations of Robotics XIV: Proceedings of the Fourteenth Workshop on the Algorithmic Foundations of Robotics 14*. Springer, 2021, pp. 294–312.

[5] Y. Zhou, H. Hu, Y. Liu, and Z. Ding, "Collision and deadlock avoidance in multirobot systems: A distributed approach," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 7, pp. 1712–1726, 2017.

[6] L. Wang, A. D. Ames, and M. Egerstedt, "Safety barrier certificates for collisions-free multirobot systems," *IEEE Transactions on Robotics*, vol. 33, no. 3, pp. 661–674, 2017.

[7] D. R. Kuhn, R. N. Kacker, and Y. Lei, *Introduction to combinatorial testing*. CRC press, 2013.

[8] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, 2005, pp. 273–282.

[9] 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.

[10] T.-D. B. Le, D. Lo, C. Le Goues, and L. Grunske, "A learning-to-rank based fault localization approach using likely invariants," in *Proceedings of the 25th international symposium on software testing and analysis*, 2016, pp. 177–188.

[11] X. Li and L. Zhang, "Transforming programs and tests in tandem for fault localization," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, pp. 1–30, 2017.

[12] X. Li, W. Li, Y. Zhang, and L. Zhang, "Deepfl: Integrating multiple fault diagnosis dimensions for deep fault localization," in *Proceedings of the 28th ACM SIGSOFT international symposium on software testing and analysis*, 2019, pp. 169–180.

[13] Y. Li, S. Wang, and T. Nguyen, "Fault localization with code coverage representation learning," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 661–673.

[14] E. Bartocci, N. Manjunath, L. Mariani, C. Mateis, and D. Ničković, "Automatic failure explanation in cps models," in *International Conference on Software Engineering and Formal Methods*. Springer, 2019, pp. 69–86.

[15] T. Yamaguchi, B. Hoxha, D. Prokhorov, and J. V. Deshmukh, "Specification-guided software fault localization for autonomous mobile systems," in *2020 18th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*. IEEE, 2020, pp. 1–12.

[16] D. Ničković, X. Qin, T. Ferrere, C. Mateis, and J. Deshmukh, "Specifying and detecting temporal patterns with shape expressions," *International Journal on Software Tools for Technology Transfer*, vol. 23, no. 4, pp. 565–577, 2021.

[17] M. Ergurtuna and E. A. Gol, "An efficient formula synthesis method

with past signal temporal logic," *IFAC-PapersOnLine*, vol. 52, no. 11, pp. 43–48, 2019.

[18] K. Choi, J. Yi, C. Park, and S. Yoon, "Deep learning for anomaly detection in time-series data: review, analysis, and guidelines," *IEEE Access*, vol. 9, pp. 120 043–120 065, 2021.

[19] H. Zhao, Y. Wang, J. Duan, C. Huang, D. Cao, Y. Tong, B. Xu, J. Bai, J. Tong, and Q. Zhang, "Multivariate time-series anomaly detection via graph attention network," in *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 841–850.

[20] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 387–395.

[21] N. Ding, H. Ma, H. Gao, Y. Ma, and G. Tan, "Real-time anomaly detection based on long short-term memory and gaussian mixture model," *Computers & Electrical Engineering*, vol. 79, p. 106458, 2019.

[22] T. Wen and R. Keyes, "Time series anomaly detection using convolutional neural networks and transfer learning," *arXiv preprint arXiv:1905.13628*, 2019.

[23] Z. Chen, D. Chen, X. Zhang, Z. Yuan, and X. Cheng, "Learning graph structures with transformer for multivariate time-series anomaly detection in iot," *IEEE Internet of Things Journal*, vol. 9, no. 12, pp. 9179–9189, 2021.

[24] R. Alur and T. A. Henzinger, "Logics and models of real time: A survey," in *Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems)*. Springer, 1991, pp. 74–106.

[25] 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.

[26] K. Sarkar, C. J. Colbourn, A. De Bonis, and U. Vaccaro, "Partial covering arrays: algorithms and asymptotics," *Theory of Computing Systems*, vol. 62, pp. 1470–1489, 2018.

[27] N. I. of Standards and Technology, "Covering array tables," 2008. [Online]. Available: https://math.nist.gov/coveringarrays/ipof/tables/table.3.2.html

[28] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE transactions on software engineering*, vol. 30, no. 6, pp. 418–421, 2004.

[29] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 167–170.

[30] R. C. Technology, "Sherpa – an efficient and robust optimization/search algorithm," in *WP-1023*. Red Cedar Technology, 2008, pp. 1–3.

[31] J. H. Ward Jr, "Hierarchical grouping to optimize an objective function," *Journal of the American statistical association*, vol. 58, no. 301, pp. 236–244, 1963.

[32] M. S. Sorower, "A literature survey on algorithms for multi-label learning," *Oregon State University, Corvallis*, vol. 18, pp. 1–25, 2010.