Automatic Optimization of Tolerance Ranges for Model-Driven Runtime State Identification

Sabine Sint, *Student Member, IEEE*, Alexandra Mazak-Huemer, *Member, IEEE*, Martin Eisenberg, *Student Member, IEEE*, Daniel Waghubinger, and Manuel Wimmer, *Member, IEEE*

***Abstract*— For continuously checking and updating the virtual representation of a real system during operation, the continuous sensing and interpretation of raw sensor data is a must. The challenge is to bundle sensor value streams (e.g., from IoT networks) and aggregate them to a higher logical state level to enable process-oriented viewpoints and to handle uncertainties about sensor measurements and state realization precision. To address these uncertainties, so-called "tolerance ranges" must be defined in which logical states are detected during operation with acceptable deviations. Specifying such tolerance ranges manually is a time-consuming, error-prone task and often not feasible due to the huge associated value search space. To tackle this challenge, the problem is turned into an optimization problem in this paper. For this purpose, we present a framework based on meta-heuristic search that enables the automatic configuration of tolerance ranges based on available execution traces of multiple sensor value streams. An exploratory study evaluates the approach. For this purpose, we implemented a lab-sized demonstrator of a five-axis grip arm robot, which we continuously monitored during operation in a simulated environment. The evaluation shows the advantage of using meta-heuristic optimizers such as Harmony Search or Genetic Algorithm to identify stable tolerance ranges automatically for state detection at runtime.**

***Note to Practitioners*—Monitoring sensor values streams is nowadays a frequently employed technique in many automation domains. However, combining and mapping single value streams to higher-level state-based representations such as state machines or other design-time related models is a major challenge due to measurement and realization precision uncertainties. Thus, simply mapping monitored raw data to these design descriptions can lead to falsely identified or missed states. To improve this situation, we present an approach that provides a mechanism to continuously analyze data streams during operation by automatically finding appropriate tolerance ranges to detect realized system states. The approach uses a small set of annotated execution traces and meta-heuristic searchers to derive optimal tolerance ranges, which provide high correctness and completeness of the identified system states. This approach represents the basis for building a "vertical bridge" from the operation technology layer considering pure sensor data streams to the IT layer where state-based process views are provided to perform monitoring and analytics, e.g., by using process mining.**

***Index Terms*— Uncertainties, meta-heuristic search, harmony search, genetic algorithm, runtime monitoring, model-driven engineering.**

I. Introduction

FOR engineering and operating software-intensive systems, precisely identifying, representing, and reasoning about runtime states of systems is a must for realizing smart system features [1], [2]. A particular challenge for effectively performing these activities, especially for successfully identifying runtime states, is the quantification of realization precision and measurement uncertainties [3]. Currently, the main focus of uncertainty quantification is on 3D-CAD models representing physical parameters during the design phase of systems [4], [5], [6]. In contrast, models that depict the logical structure and behavior of the system in operation are often not virtually mapped and analyzed [7].

In this paper, we target state-based monitoring, focusing on the logical behavior of the systems. In particular, we collect sensor data streams of an actual system, e.g., running a robot on the shop floor, and semantically lift it to a higher level of abstraction, the so-called "logical state level". In previous work [8], we took the first step by presenting a framework to automatically identify system states during operation. This framework enables continuous tracing of components of a software model (e.g., a state machine) based on sensor value streams gathered at the system level. However, lessons learned show that the state identification task is challenging due to uncertainties concerning the measurement of data and realization precision of system components. Therefore, we consider "tolerance ranges" to optimize the state detection process. A *tolerance* is the deviation of a quantity from the standard dimension (normal state) that does not negatively affect the functionality of a system or its components [9]. The consideration of tolerance ranges is needed since there are sensor delays and inaccuracies when measuring, and only a certain degree of state realization precision can be reached, which hinders the identification of exact values as they were initially set in the corresponding software model at design time. Defining such ranges manually, as we did in our previous work [8], is an error-prone and time-consuming task due to the huge search space for finding optimal parameters. Please note that

Manuscript received 24 January 2024; accepted 20 March 2024. This article was recommended for publication by Associate Editor E. Estevez and Editor B. Vogel-Heuser upon evaluation of the reviewers' comments. This work was supported in part by the Austrian Federal Ministry for Digital and Economic Affairs; and in part by the National Foundation for Research, Technology and Development (CDG). *(Corresponding author: Sabine Sint.)*

The authors are with CDL-MINT, Department of Business Informatics and Software Engineering, Johannes Kepler University Linz, 4040 Linz, Austria (e-mail: sabine.sint@jku.at).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TASE.2024.3386313.

Digital Object Identifier 10.1109/TASE.2024.3386313

statistical methods that estimate deviations in sensor data [10] lack particular support for the state detection task as we are investigating. This includes the significance of individual sensors and the importance of specific deviations in specific circumstances when distinguishing the states.

In this paper, we propose integrating meta-heuristic search techniques into the state detection framework to calibrate the tolerance ranges for detecting states from sensor logs. In particular, we adopt two population-based algorithms, namely Harmony Search (HS) [11] and Genetic Algorithm (GA) [12], where each individual represents a possible configuration for the detection tolerances. The contributions are as follows:

- A characterization of the uncertainty problem when gathering state traces from sensor logs based on the modeled process flow. We consider exhaustive and manual methods impractical due to the complexity of the problem.
- A framework for the step-wise alignment from modeled, discrete system behavior to actual execution subject to uncertainty, logged as continuous sensor streams, to facilitate traceability and monitoring capabilities.

We perform a case study based evaluation of our approach on two different workflows using a lab-sized demonstrator of a five-axis grip arm robot. We investigate HS and GA algorithms' detection correctness and completeness compared to Random Search. Results show that meta-heuristic searchers are effective techniques for finding tolerance ranges that enable effective detection of discrete state events.

The remainder of this paper is structured as follows. Section II explains the background and motivation of our work. Section III characterizes the complexity of the investigated problem and presents meta-heuristic search as a solution for finding optimal tolerance ranges for runtime state detection. Section IV presents the evaluation of our approach by a lab-sized demonstrator. Finally, in Section V, we discuss related work and conclude with an outlook in Section VI.

II. Background

In this section, we present a motivating example and give a brief overview of previous research efforts [8], which have led us to the problem space and the solution presented in this paper. We provide a short introduction to *Harmony Search (HS)* [11] and *Genetic Algorithm (GA)* [12], both forming the basis for our meta-heuristic approach.

A. Motivating Example

Consider an automation system in the form of a three-axis grip arm robot (gripper), consisting of sensors, actuators, and a controller, operating in a lab-sized environment (cf. Fig. 1). The gripper's controller is model-driven engineered by using the *Systems Modeling Language (SysML)* [13], specifically the *Block Definition Diagram (BDD)* and the *State Machine Diagram (SM)*. However, the presented approach is neither limited to this language nor to these diagram types. There are several other languages, such as UML[1] or BPMN,[2] which are also suitable for modeling the gripper and its workflow.

[1]https://www.uml.org
[2]https://www.bpmn.org

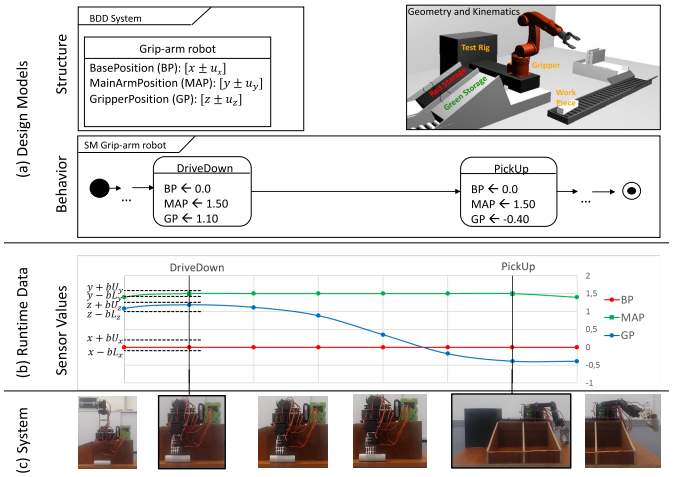


Fig. 1. Design-time (a) and runtime perspective (b) of the gripper system (c).

The BDD defines the different angle positions the gripper takes at runtime by properties of its actuators: `Base Position (BP)`, `Main Arm Position (MAP)`, and `Gripper Position (GP)` (cf. Fig. 1a, `BDD System`) and their values (cf. Fig. 1a, `x,y,z`) for different states (e.g., `DriveDown`, `PickUp`, etc.). For instance, when driving down, `MAP` is set to 1.50 (cf. Fig. 1a, `SM Grip-arm robot`) in the design model. Such value assignments are mainly based on the machine operator's knowledge and the manufacturer's configuration parameters. The problem is that sensors will never measure precisely predefined values. On the contrary, they vary (upwards and downwards) from measurement to measurement. This fact is considered in Fig. 1a with the symbol $\pm$`u` ("`u`" indicates uncertainty) indexed by x, y or z for each of the properties of the BDD.

At design time, the behavior of the gripper managed by the controller is modeled by various state settings and transitions as a predefined workflow by the SM (cf. Fig. 1a, `SM Grip-arm robot`). In the given example, there are two states: `DriveDown` and `PickUp` with different value assignments for specifying the respective angle position to be taken when operating.

During operation, the gripper moves within a pick-and-place unit based on the SM (cf. Fig. 1c), and all its movements are recorded by angle sensors and returned as continuous sensor value streams of the properties (`BP, MAP, GP`) to a log recording system (cf. Fig. 1b). These records display that the gripper does not move time-discrete from one state to another, as initially modeled by the workflow in the SM. On the contrary, the movement is continuous, i.e., long-running operations instead of instant realizations.

The actual run shows that, e.g., when the gripper is in the state `PickUp`, the initially (precisely) assigned values (`BP` $= 0.0$, `MAP` $= 1.50$, and `GP` $= -0.40$) are not reached. For monitoring a system, it is, therefore, essential to identify if the measured state values have acceptable degrees of variation or if the system must be reconfigured. For this purpose, appropriate *tolerance ranges* (i.e., lower and upper bounds) are needed when recording sensor values, as depicted in Fig. 1b, e.g., see the green curve: $y + bU_y, y - bL_y$. Without such ranges, e.g.,

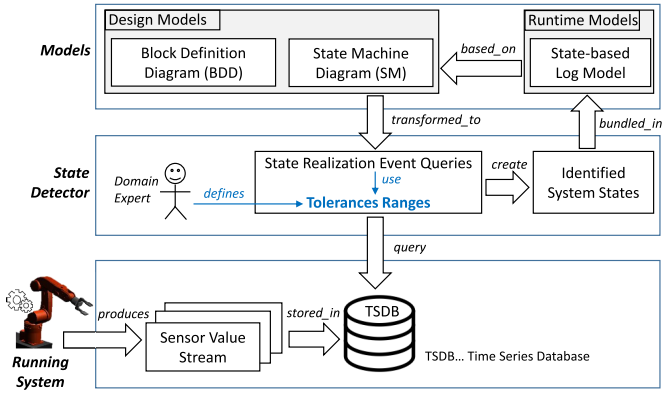


Fig. 2. Architecture for model-driven runtime state identification based on [8].

a useful target-actual-comparison of models at design time (Fig. 1a) with its real-world counterparts (Fig. 1c) cannot be performed. This is needed, e.g., when improving the quality management during the system's monitoring process to determine how large the tolerances could be without (negatively) affecting other factors or the outcome of a production process.

B. Model-Driven Engineering for State Detection at Runtime

In previous work [8], we present a model-driven runtime state identification approach where we derive state realization event queries from design models. We listen to sensor value streams during runtime to detect system states. As illustrated in Fig. 2, we automatically transform a `State Machine Diagram` into `State Realization Event Queries`, which uses manually defined `Tolerance Ranges`, to match `Sensor Value Streams` with predefined variable values coming from the design models. The sensor streams are collected at runtime in a `Time Series Database (TSDB)`. We execute the event queries on the TSDB, which result in `Identified System States`, bundled into a `State-based Log Model`. Based on this log, analysis techniques, e.g., [14], can be applied to validate whether the runtime model corresponds to the design time model.

Lessons learned showed that to detect the states successfully, an expensive manual calibration process is required to set tolerance ranges properly. Such a process consists of the following steps: (i) checking the results after each iteration, (ii) manually adjusting the ranges each time, (iii) verifying if the achieved results are more precise and complete results in the next iteration, and so on. To handle the enormous search space, we treated the adjustments for all angle sensors similarly (± 0.1 as tolerance range equally for all sensors). Such a "one-size-fits-all" approach is inappropriate for practical use, and the manual configuration is too time-consuming and error-prone. The outcomes made clear that even if such a trial & error approach may work for small-scale systems, it is of exponential complexity in large-scale systems as further discussed in Section III-B.

C. Harmony Search (HS) Algorithm

The HS algorithm was introduced as a meta-heuristic algorithm for optimization tasks [11]. Over the years, numerous HS variants, HS comparisons to other heuristic techniques, and applications of HS in science and industry [15] have been presented.

1) Analogy: The name and idea grounding the functionality of HS originate from the domain of musical improvisation [11]. It is based on the process of composing a novel harmony. In this process, the musician has three options: (i) including notes of music pieces learned by heart, (ii) adding adjusted versions of notes that are remembered, and (iii) adding random notes. This process of composing harmonics creates new harmonies successively by aiming to find the most audibly pleasing one. Thus, the process can be seen as an optimization task [15], where all notes the musician can select represent a search space. Each iteration during composing yields a solution vector (i.e., ordered notes of a certain harmony) that is then evaluated by an objective fitness function where under-fitting parts will be rejected, and the fitting ones will be stored. Thereby, the fitness of the population, i.e., harmony memory, gradually increases w.r.t. the objective function.

Algorithm 1 Harmony Search [11], [15]

1: Define objective function $f(x), x = (x_1, x_2, \ldots, x_d)^T$
2: Initialize HMCR, PAR, BW, HMS
3: Initialize Harmony memory with random harmonies
4: **while** ($t <$ max number of iterations)
5: **while** ($i <$ number of decision variables)
6: **if** (rand $<$ HMCR), Choose a value from HM for the variable i
7: **if** (rand $<$ PAR), Adjust the value by adding a certain amount
8: **end if**
9: **else** Choose a random value
10: **end if**
11: **end while**
12: Accept the new harmony (solution) if better
13: **end while**
14: Return the current best solution

2) Procedure: The procedure of the HS algorithm is shown in Algorithm 1. At first, the search-defining parameters are set, and the *Harmony Memory (HM)* is initialized [lines 1-3]. In the next step, new solution candidates are explored within the search space [lines 4-13]. Each iteration results in a solution vector to be evaluated and used as a substitute for the worst solution stored in the HM. This iterative process continues until a certain stop criterion is met. Such a criterion may be a fixed number of iterations, or a solution that satisfactorily meets the expected outcome (fulfilling a certain threshold) of the objective function is found.

3) Parameters: HS builds on four parameters [11]:

- *Harmony Memory Consideration Rate (HMCR)*: the probability of randomly choosing a solution from the HM as a new solution candidate.
- *Pitch Adjusting Rate (PAR)*: the probability of adjusting the solution randomly picked from the memory (assuming that a solution is reused based on HMCR).

- *Bandwidth (BW)*: value range from which a value is randomly selected for optimizing the solution vector (assuming that a solution from HM is reused based on the HMCR and adjusted based on PAR).
- *Harmony Memory Size (HMS)*: the number of solution vectors stored in the HM.

Depending on the objective function, a minimum and maximum value defines the search space. This avoids the consideration of candidates that are far off. As discussed in [16], there is a trade-off between diversification and intensification for any meta-heuristic approach. For instance, PAR, in combination with BW, controls the search intensity of the neighborhood of previously considered reasonable solutions. When setting HMCR too low, the possibility of finding local optima is strongly reduced since solutions from memory are rarely exploited for searching for new candidates. In contrast, if HMCR is too high (i.e., solutions are mostly reused from memory and adjusted slightly), the risk increases that parts of the search space will not be considered, and thus, potential optima will never be found.

D. Genetic Algorithm (GA)

First conceptual considerations on the GA go back to the work of Holland [12]. Further developments and features led to versatile adoptions [17], and thus, GAs are successfully applied in several application areas [18].

1) Analogy: GAs belong to the family of evolutionary algorithms [17] and are stochastic, population-based algorithms that reflect the evolution process. Within a population, new individuals emerge as offspring from the current generation, and only the fittest individuals survive to form the next generation's population, and so on. This evolutionary cycle is characterized by selecting individuals for reproduction, the reproduction process, and the fitness assessment [17]. In GAs, an individual is represented as a chromosome consisting of a set of genes, which encode the decision variables of an optimization problem. Furthermore, reproduction is considered by selecting chromosomes and applying crossover and mutation operators.

Algorithm 2 Genetic Algorithm (based on [18])

1: Initialize $P(0)$, p_c, p_m, $t = 0$
2: Evaluate $P(0)$
3: **while** $(notTerminated)$
4: $P_O = Selection(P(t))$
5: $P_O = Alter(P_O)$ // crossover and mutation based on p_c, p_m
6: Evaluate P_O
7: $P(t+1) = Selection(P(t), P_O)$
8: $t = t + 1$
9: **end while**
10: Return the current best solution

2) Procedure: Algorithm 2 shows the general steps followed by GAs [18]. During initialization, an initial population $P(0)$, which may comprise random individuals, is generated and evaluated, and crossover (p_c) and mutation (p_m) probabilities are set [lines 1-2]. The evolutionary cycle [lines 4-8] is bound to a termination condition, e.g., a fixed number of generations. Therein, the offspring population P_O is selected from the current population. New individuals are then produced as the result of recombination and mutation applied to individuals of P_O. Individuals carried on to the new population $P(t+1)$ are ultimately selected from the original population of the current cycle and the altered offspring population.

3) Parameters: The following operators are typically used:

- *Crossover*: Offspring are produced by recombining two individuals during the crossover. A common approach is to exchange parts based on one or multiple cut points.
- *Mutation*: Mutation performs a variation on the individual level to ensure genetic diversity after crossover. Such can be achieved, e.g., by reversing the gene sequence of a chromosome or displacement of genes.
- *Selection*: Subject to evolutionary progress, only the fittest individuals are carried on in the population considering the current generation and the emerged offspring.

Another crucial factor is the encoding scheme used [18], and the choice of operator also depends on the representation of a solution. In this respect, recombination and mutation strategies exist specifically for, e.g., the binary, permutation, and real-valued encoding schemes. For floating-point representation, commonly, the mutation operator changes the gene within its domain given by the bounds, whereas for recombination, values can be merely exchanged (discrete recombination), or the intermediate point between the parents is taken (intermediate recombination) [17].

III. Approach

We now tackle the aforementioned challenges by applying a meta-heuristic search approach to find optimal tolerance ranges for state identification, starting from the MDE-based approach of our previous research [8].

A. Overview

Fig. 3 gives an overview of our approach, which we describe in more detail in Section III-C. The input for the optimizer is two-fold: (i) the `value streams` of each sensor and (ii) a `reference set` of annotated logical states. This set is derived through manual annotation of the sensor measurements, e.g., by a system operator, after keeping records of the execution workflow. Based on these inputs, the `meta-heuristic algorithm` searches for a configuration that best reflects the discrepancy between the model and reality of the state properties. By instantiating one of the solution configurations, which are continually adjusted, it follows a matching process between the reference set and the states captured. For this purpose, the optimizer considers two objective functions: (i) `maximize (F-measure)`, i.e., the harmonic mean of precision and recall of the detection process, and (ii) `minimize` / `maximize (distance between offsets)` for guiding the search process. As output, we get `Optimized Offsets (OpOs)`, which are the optimal lower (oL_{s_i}) and upper offsets (oU_{s_i}) to derive the optimal

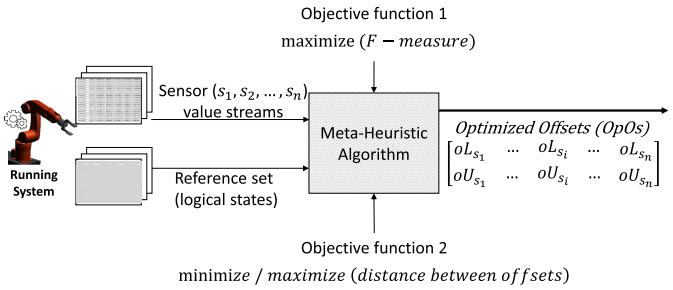


Fig. 3. Approach overview: Computation of optimized tolerance ranges.

tolerance range for each state's property for a sensor (s_i). This enables identifying and verifying the optimal tolerance ranges for each property of a system operating within a certain setting.

Before we present our approach in detail, we briefly discuss why a meta-heuristic search is needed for the given problem.

B. Problem Complexity

Due to the consideration of float numbers for the property values of a system and the combinatorial complexity of value combinations, the search space can grow very quickly. Thus, finding the best combination of tolerance ranges (i.e., lower and upper offsets) for each sensor is computationally intensive. In particular, the search space is considered as a variation problem, as shown by the following equation: where k is the number of possibilities of objects (i.e., number of options for upper and lower offsets) and n the number of places (i.e., number of sensors).

$$SearchSpace = (k^2)^n \tag{1}$$

Considering this magnitude, not only is a manual search impossible, but even an exhaustive automated search. Thus, we need an intelligent search algorithm to find good tolerance ranges in a reasonable time.

C. Meta-Heuristics for Tolerance Range Identification

We now describe how the meta-heuristic approach is applied to the discussed tolerance range optimization problem. For this purpose, we use the lab-sized gripper introduced before as an example. As a starting point, we take a set of deployed sensors $S_{sensors}$ of the system, e.g., the gripper, from which we aim to abstract a logical state model based on the sensor value streams gathered during operation. We assume that inaccuracies (e.g., based on measurement uncertainty) exist in the sensor readings and/or recordings, as is often the case in practice [19]. Additionally, we have to consider that such inaccuracies are not symmetrical in both directions (upwards and downwards). Therefore, we need for each sensor of the set ($s_i \in S_{sensors}$) a so-called *upper offset* (oU_{s_i}) and *lower offset* (oL_{s_i}). These optimized offsets lead to the optimal tolerance ranges for the sensor properties for the sensors ($S_{sensors}$) considered in each state:

$$OpOs(S_{sensors}) = \begin{bmatrix} oLs_1 & oLs_2 & \cdots & oLs_n \\ oUs_1 & oUs_2 & \cdots & oUs_n \end{bmatrix} \tag{2}$$

As already mentioned in Section III-A, our meta-heuristic approach uses two sets as input to compute a set of *Optimized Offsets (OpOs)* as output (cf. Fig. 3). One set is the sensor value streams of the system logged as tuples $(t, s_1, \ldots, s_n)$ with timestamp t during operation, and the other set is the reference set, a subset of the logged value streams manually annotated with logical state information. For the gripper example, the value streams come from the individual angle sensors. In this context, identifying states from continuous value streams can be seen as an information retrieval problem. Given the streams recorded over a period, only those tuples that correspond to a point in time when a specific state occurred, as indicated by a match with the reference set, should be retrieved. Which tuples are retrieved thereby depends on the state realization event query, where upper/lower offsets are added/subtracted to the state's properties defined in the behavior model [8]. In this regard, we denote optimal offsets as the ones that lead to a consistent and complete reconstruction of the states in the reference set. The challenge now is to find the sweet spot for the permitted deviation: On the one hand, large enough to allow for regular deviations, on the other hand, small enough that the resulting range for one state allows for its identification and differentiation from another. However, further circumstances in reality may threaten the consistency of the identification. For example, the gripper holding out at the same position for an extended time period further complicates the detection. In these dwell times, the sensor streams will be recorded with identical properties over a certain time window, while in the reference set, states of the executed workflow are noted once, namely when the corresponding position is reached for the first time. Accordingly, in such cases, the query erroneously leads to multiple detections of the same state. In a further experiment in Section IV, we account for this by temporally ordering the set of detected states, then considering only the first instance of each immediate duplicate occurrence.

The precision determines which state identifications were correct, whereas the recall determines which proportions of the actual states to detect were identified [20]. Both metrics are, by definition, in a field of tension since improving precision typically reduces recall and vice versa. In terms of our approach, the best set of tolerances is the one that enables the recognition of all logical states without missing any or providing false positives. For the evaluation of one tolerance range configuration, both measures are calculated for each state st and respectively combined to their harmonic mean, the so-called F-measure, denoted as $F_{st}(O)$ in Equ. 3.

$$F_{st}(O) = 2 * \frac{Precision_{st}(O) * Recall_{st}(O)}{Precision_{st}(O) + Recall_{st}(O)} \tag{3}$$

To obtain reasonable offset values, the first objective function for the meta-heuristic algorithm in Equ. 4 aims to maximize the arithmetic mean ($F_{mean}(O)$) over the sum of all F-measure values ($F_{st}(O)$) for all states $st_j \in S_{states}$.

$$max\ F_{mean}(O) = \frac{1}{|S_{states}|} \sum_{j=1}^{S_{states}} F_{st_j}(O) \tag{4}$$

The second function serves to optimize the tolerance limits so that the tolerance ranges turn out as small/large as possible:

$$min/max\ O_{sum} = \sum_{i=1}^{S_{sensors}} (oL_{s_i} + oU_{s_i}) \tag{5}$$

The second objective function can be applied to two different strategies. By limiting the acceptable tolerance ranges to a minimum, measurement accuracy deterioration (equipment aging, errors, etc.) can be detected immediately. This provides the opportunity to investigate the causes early and take appropriate countermeasures. However, if the primary purpose is to detect states as accurately as possible in the long term, maximizing the deviation ranges seems reasonable to allow potentially higher deviations. This is necessary, e.g., when improving quality management during system monitoring to determine how large tolerances can be without (negatively) affecting other factors or the outcome of a production process.

In the search process, the meta-heuristic algorithms produce new solution vectors (cf. Definition 2) based on their underlying mechanisms. These vectors are then employed to perform state detection, the detected states of which are used to evaluate the solution against our objective functions under consideration of real occurrences given in the reference set.

IV. Evaluation

We now present the setup and the results of an exploratory study according to the guidelines of [21]. All artifacts of the evaluation, including prototypical implementation, data, computations, statistical tests, and figures, can be found in the accompanying repository.[3]

A. Research Questions

The study addresses the following research questions (RQs).

RQ1 Detection performance: How good is the state detection performance? We use HS and GA to evaluate a certain number of configurations and evaluate as many randomly generated configurations with Random Search (RS) to draw a comparison. Of interest are consistency and completeness of the detection on the set of training traces, as well as how quickly improvements occur. Moreover, how well the found offsets hold for the test set is assessed. In general, the success of search algorithms can be affected by different parameter settings. Thus, we first perform parameter tests to configure them accordingly.

RQ2 Dwell time impact: How critical are holdouts of the gripper for our approach? With the state realization queries not carrying a temporal dimension, such cases impede unambiguous identification. To this end, we investigate the consistency of recognition before and after specific treatment of successive duplicates in the recognized state sequence.

RQ3 Tolerance level optimization: How flexible are the tolerance ranges? Here, we investigate the influence of a tendency towards rather wider or narrow detection ranges. They should not come at the expense of detection performance. In particular, we are interested in whether the minimization or maximization of the tolerance range, denoted by the lower and upper offset, impacts the F-measure results. Furthermore, we aim to assess to what extent the tolerance ranges can be minimized or maximized using this objective function.

B. Study Setup

For the use cases described in the following, we have logged the sensor properties and created the reference set for 60 workflow executions each. Even if an optimal tolerance setting can be determined to reconstruct these traces, this should also apply to new executions in the running system. Therefore, for one experiment execution, we consider only ten (randomly selected) traces as a training set to determine the offset configuration and subsequently apply it to the held-out set of 50 remaining traces, i.e., the test set. By doing so, we obtain an estimate of the detection performance for the system in production while keeping the preliminary annotation overhead for the reference set low.

1) Use Cases: We re-use the example introduced in Section II-A and extend it to a five-axis gripper. We log the rotations of all angle sensors during every run using the time-series database InfluxDB.[4] We implement two use cases with different scenarios and complexity for a production unit.

Use Case 1 (UC1): The gripper takes workpieces from a conveyor belt, puts them down on a test rig, and finally releases them in a red or green storage box based on the information coded on each workpiece through a QR-code that is read by a camera on the rig. The gripper can take up to 14 logical states during operation. To ensure the plausibility of our approach, only two rather distinctive logical states, `DriveDown` and `PickUp`, are considered in a first test to determine tolerance ranges. Subsequently, tolerance ranges are computed and analyzed under consideration of all 14 logical states the gripper can take.

Use Case 2 (UC2): This use case models a simple pick-and-place unit within the working station, where a gripper takes workpieces from one conveyor belt and puts them on a second one for further processing. In this scenario, the gripper runs through 8 logical states.

We use these scenarios to evaluate our approach, once by maximizing the F-measure and minimizing/maximizing offsets and once by maximizing the F-measure only. Since HS is a single objective approach, the following procedure/weighting was implemented when using two objective functions. First, we compare whether the new solution has an F-Measure that is equal to or higher than those in the memory. If this is the case, the second objective function picks a solution where the distance between the lower and upper offset is smaller/larger.

For implementing GA, we employ Jenetics.[5] We use the same encoding and objective functions as for HS and a custom selection operator following the same hierarchical evaluation approach: optimizing the F-measure is paramount, and the size of the tolerance limits is secondary.

[3] https://github.com/cdl-mint/AutomOptStateIdentification

[4] https://www.influxdata.com

[5] https://jenetics.io

2) Evaluation Metrics: For answering our RQs, we compare our meta-heuristic approaches HS and GA with RS. The comparison with the baseline search through RS should evaluate whether a meta-heuristic search approach is necessary. The comparison of HS and GA checks whether a local or global search is more efficient for finding tolerance ranges. Therefore, we analyze the F-measure of the different searches and apply statistical tests to validate our assumptions. Meta-heuristic algorithms are stochastic optimizers that may produce different results for the same input. Thus, for each configuration (i.e., different scenarios of the two use cases), we perform 30 independent runs for every problem instance. Since our samples have the same underlying population (where we do not know the distribution), we apply a non-parametric statistical test, namely, the Wilcoxon rank-sum test [22] for two independent samples. We use R[6] to compute the tests with a significance level of $\alpha = 5\%$. It indicates the probability of rejecting H0 if it is true. For evaluating the performance, we use a one-sided Wilcoxon rank-sum test to show whether there is a statistically verifiable significance between HS, RS, and GA.

3) Parameter Settings of HS and GA: Given the variety of problem kinds, encodings, performance metrics, and operators, general recommendations for any problem type are considered infeasible [17], [23]. Settings that have proven effective for certain applications nevertheless provide guidance. In this respect, for GA, ranges $p_c = .6 - .95$ and $p_m = .001 - .01$, and population sizes between 20 and 100 have shown moderately different results on the same test suite [23]. For real-valued encodings, a common setting is .7 (p_c), .05 (p_m), and 10 for the population size [17]. For HS, common values range between .7-.95 and .1-.5 for HMCR and PAR, respectively [15], [16]. Regarding BW, a general recommendation is to take 1% to 10% of the value range of the variables' domain [24].

In order to exclude the setting as a possible determining factor for the final results, we dedicate effort towards testing different parameter configurations before performing the actual experiments. In this sense, we start from a canonical configuration for HS and GA and proceed one at a time. This means that we first vary only the population size, then proceed to determine the selection and mutation operators, and finally test different crossover and mutation rates. Likewise, different memory sizes are tested before the control parameters of HS are varied. In each step, we advance with the configuration showing the best mean fitness at the end of the search. The results shown in Table I are laid out in Section IV-C.

The termination criterion plays a crucial role in the fair and unbiased comparison of algorithms [25]. Note that the bottleneck in our case is the evaluation of each tolerance setting by employing it for state detection. Accordingly, for each new candidate solution, a query is performed in the TSDB for each state. Therefore, adhering to the recommendation in [25], the same number of objective function evaluations is granted for each execution of the algorithms after evaluating the solutions in the initial population.

[6]https://www.r-project.org

TABLE I
PARAMETER SETTINGS OF HS AND GA FOR EVALUATION

GA	HS
Population size: 25 Offspring fraction (O_f) = .6 Self-adaptive Gaussian mutation [17] (p_m) = .05 Single-point crossover (p_c)= .8 Tournament selection (n) = 3	Memory size: 50 HMCR = .8 PAR = .3 BW = .007

C. Results and Interpretation

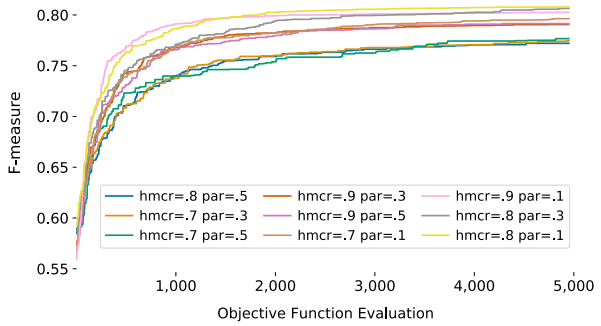
We now present our results to answer the RQs.

1) Results RQ1: We measure the performance of our approach to determine the efficiency based on the objective function evaluations for each algorithm, i.e., HS, GA, and RS. Additionally, we evaluate the effectiveness by validating the ability to produce desired results and, finally, determine the quality of achieving these results by computing an F-measure value for each use case. This is done based on the best configuration, which is determined below.

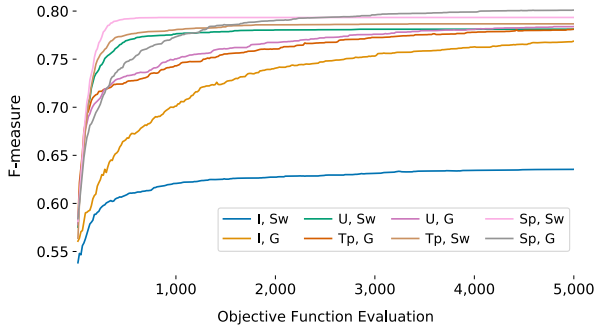
The parameter evaluation is performed as described in Subsection IV-B3 starting from a common application, namely: Uniform crossover ($p_c = .8$) and self-adaptive Gaussian mutation [17] ($p_m = .1$) for GA and $HMCR = .9$, $PAR = .1$, $BW = .07$ for HS. The population size shows better performances on the lower side ($N_p \leq 100$), whereas marginal differences are observable for HS and tested memory sizes. Regarding HMCR and PAR in Fig. 4a, extensive recycling of memory solutions ($HMCR = .8 - .9$) and the occasional variation of a reused value ($PAR = .1 - .3$) seems more beneficial than focusing more on random exploration and a higher probability of varying more values of a tolerance configuration at the same time. Slight differences are observable for BW set between 1% to 10% of the total tolerance range with a tendency towards higher improvements through more (higher PAR) and smaller (lower BW) adjustments. Considering different alteration operators in Fig. 4b, remarkably, intermediate approaches are outperformed by the generically applicable discrete recombination [17] approaches such as uniform or single-point crossover. In this respect, assembling a new configuration from the tolerance settings of two configurations appears superior to their interpolation. Furthermore, swapping genes in the gene sequence using Swap mutation [17] performs similarly to using Gaussian mutation, which applies to floating point encodings. That is, exchanging the tolerance values between sensors seems as reasonable as preserving and slightly adapting them. In their application, small differences occur, with the best performance emerging from a regular crossover and a moderate mutation probability ($p_c = .8$, $p_m = .05$). Ultimately, we use the best configurations (cf. Table I) to perform the experiments for our case studies.

For a sanity check, we limit UC1 to two logical states. Results showed no significant difference regarding effectiveness when computing optimized offsets, whether using HS, GA, or RS. All algorithms achieve F-measure values of 1.

Considering the complete workflow of UC1, Fig. 5a shows the mean F-measure scores of the different algorithms for the

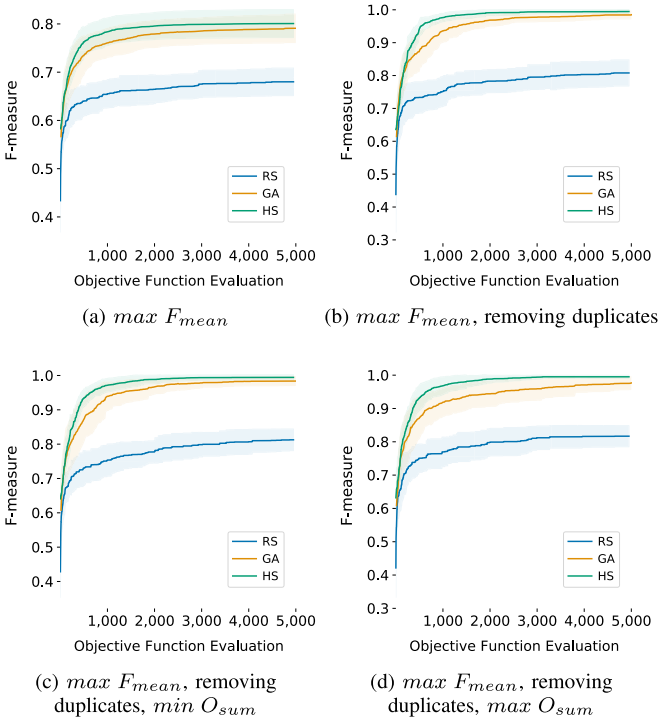


(a) HS: Varying control parameters HMCR and PAR



(b) GA: Varying crossover (**I**ntermediate, **U**niform, **S**ingle-**p**oint, **T**wo-**p**oint) and mutation (Self-adaptive **G**aussian, **Sw**ap) operators

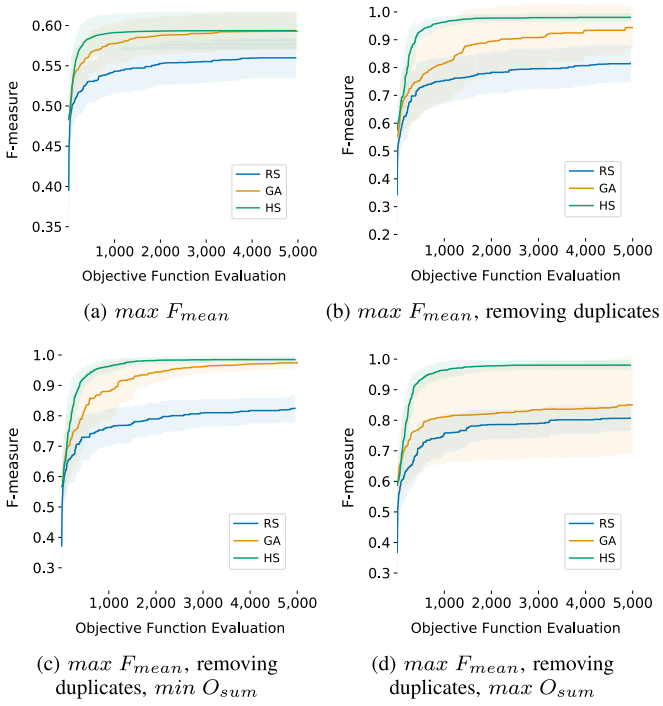
Fig. 4. Varying control parameters for HS and GA.



(a) $max\ F_{mean}$ (b) $max\ F_{mean}$, removing duplicates

(c) $max\ F_{mean}$, removing duplicates, $min\ O_{sum}$ (d) $max\ F_{mean}$, removing duplicates, $max\ O_{sum}$

Fig. 5. UC 1: Mean F-measure development (30 independent runs). Shaded areas indicate the 95% confidence interval for the true mean.



(a) $max\ F_{mean}$ (b) $max\ F_{mean}$, removing duplicates

(c) $max\ F_{mean}$, removing duplicates, $min\ O_{sum}$ (d) $max\ F_{mean}$, removing duplicates, $max\ O_{sum}$

Fig. 6. UC 2: Mean F-measure development (30 independent runs). Shaded areas indicate the 95% confidence interval for the true mean.

training set. It is apparent that RS (blue line) lags behind HS (green line) and GA (orange line), with HS seeming to deliver slightly better results than GA. The one-tailed Wilcoxon rank sum test confirms these observations and shows that the final F-measure values of HS are higher than those of GA ($p = .025$). This result is also observed when comparing the precision; HS finds the most relevant elements, and HS and GA are both better compared to RS ($p < .01$, for all hypotheses). Concerning recall, there is no significant difference between the three algorithms. Regarding the test set, the picture is similar for all metrics: RS is worse than the meta-heuristics regarding F-measure and precision, and there are no significant differences for recall. When comparing HS and GA, there is no significant difference in F-measure and recall, whereas, for precision, the test shows that HS is slightly better than GA ($p = .045$).

In UC2, with 8 logical states, the F-measure score of RS also lags behind the scores of the meta-heuristic searchers (cf. Fig. 6a, blue line). This result is also confirmed by the statistical tests. Again, there is no significant difference between the algorithms in terms of recall. Notably, there is also no significant difference in the metrics (F-measure, precision, recall) between HS and GA. This is already indicated by Fig. 6a. In this evaluation configuration, we get a maximum mean F-measure of .594 for HS (compared to .593 and .564 for GA and RS, respectively).

Table II gives a detailed list of the mean values and standard deviations for F-measure, precision, and recall of the training and test sets for UC1 and UC2. In all cases, precision is the decisive factor for low F-measure values.

2) Interpretation RQ1: The statistical tests indicate that HS and GA yield better results than RS in both use cases. All algorithms achieve good results (high recall) regarding recall of relevant items, thus, few false negatives. However, the rate of false positives is high, as seen from the low precision.

TABLE II
EVALUATION METRICS (MEAN AND STANDARD DEVIATION) ON TRAINING AND TEST SET

Case	Alg.	Training set						Test set					
		F-measure		Precision		Recall		F-measure		Precision		Recall	
		$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
UC1	HS	.801	.029	.713	.030	.995	.005	.759	.016	.666	.017	.982	.008
	GA	.792	.028	.701	.029	.996	.005	.755	.016	.659	.018	.985	.008
	RS	.686	.028	.566	.027	.993	.009	.655	.027	.532	.029	.994	.005
UC2	HS	.594	.023	.474	.026	.989	.012	.576	.008	.455	.008	.974	.012
	GA	.593	.023	.475	.027	.989	.012	.574	.006	.452	.007	.975	.012
	RS	.564	.023	.442	.025	.986	.012	.555	.013	.430	.014	.980	.014

TABLE III
EVALUATION METRICS (MEAN AND STANDARD DEVIATION) ON TRAINING AND TEST SET - WITH FILTERING DUPLICATE DETECTIONS

Case	Alg.	Training set						Test set					
		F-measure		Precision		Recall		F-measure		Precision		Recall	
		$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
UC1	HS	.995	.006	.995	.006	.995	.007	.967	.011	.969	.011	.966	.011
	GA	.986	.011	.988	.011	.985	.012	.963	.012	.965	.012	.960	.013
	RS	.809	.039	.775	.052	.882	.042	.790	.045	.757	.055	.876	.041
UC2	HS	.980	.014	.984	.012	.977	.017	.942	.021	.944	.022	.939	.020
	GA	.945	.076	.944	.090	.954	.036	.914	.074	.913	.032	.923	.021
	RS	.815	.065	.817	.066	.814	.065	.798	.059	.799	.059	.797	.060

Hence, the occurrence of some states is perceived too often in contrast to their actual occurrences.

On the one hand, the reason for this indeed lies in the underlying sensor logs, which show identical values over periods where the gripper arm persists longer at specific points in the sequence. Here, there are inevitably multiple detections for the single occurrence of a state. On the other hand, the distinctness of the states affects the tolerance ranges to identify them unambiguously.

Taking UC2, where the states are very closely located, no significant difference was observed between HS and GA. In contrast, HS provided marginally better results for UC1, where the states are less similar. This suggests that the functioning of HS, which is mainly based on locally oriented and subtle changes, favors the search for functioning tolerance areas. Conversely, changes in the sense of a more globally oriented search, as with GA, could be too diverse and thus rather detrimental to the convergence of the individual sensors to the proper ranges.

The comparison with RS reveals that efficiently and effectively computing lower and upper offsets can only be achieved using an intelligent search, such as a meta-heuristic optimizer.

3) Results RQ2: Based on the knowledge gained from RQ1 and the question of the influence of dwell time, we investigate the influence of a filter for detecting consecutive duplicates in the detected state sequence.

In UC1, with 14 logical states, Fig. 5b shows the mean F-measure scores of the different algorithms for the training set with filter. It indicates that the detection performance can be significantly improved compared to the approach without duplicate filters (cf. Fig. 5a). This is also evident from the statistical comparison of F-measure and precision for all algorithms. Regarding recall, there is no significant difference between GA and RS algorithms with and without filters. The recall of HS is a shade below the values without the filter. When comparing the algorithm performances with the filter approach, we find that HS and GA return better F-measure values than RS (cf. Fig. 5b). Statistical tests also confirm this result. Furthermore, HS and GA outperform RS in all metrics. For the training set, the tests also show that HS is better than GA, while for the test set, this is only marginally the case for recall, and otherwise, both perform equally well.

For UC2, performance can be improved with filtering as well, as can be seen in Fig. 6b. The tests show the same picture as for UC1—precision and F-measure are significantly better. For recall, the detection is lower than without using the filter approach. Comparing the results from the filtered alternatives, the picture that emerges in Fig. 6b also materializes statistically: HS is one step better than GA, and both clearly outperform RS.

Table III gives a detailed list of the mean values and standard deviations for F-measure, precision, and recall of the training and test sets for UC1 and UC2 with the filtered version. The filter makes it possible to improve significantly precision, which has a strong overall positive effect on state detection performance.

4) Interpretation RQ2: The results show that detection works substantially better when the effects of dwell time can be mitigated, in this case, by filtering out duplicate detections. A closer look at the detected and actual states reveals the reason for the lower precision without filter: There are several false positives since a single state change was identified multiple times. This occurs when the gripper moves slowly or is stationary, and therefore, values from the streams are within the offsets for multiple sensor ticks. Therefore, it is clear that the temporal aspect plays an essential role in the practical reconstruction of the workflow on an abstract-logical level. However, this is not considered in the creation of the logs or the retrieval queries. It can be seen that with the filtering approach as described in Section III-C, the recurrences in the logs can only be masked out to a large extent, as the substantially increased precision demonstrates.

Regarding the comparison of the searches, our experiments show that the meta-heuristic approaches perform better than RS and give better results. Moreover, a local search with the filter seems superior to a global searcher as HS often yields significantly higher results than GA, especially when sensor values from different states are similar, as is the case with UC2.

5) Results RQ3: Comparing the optimization of F-measure alone and additional optimization of the tolerance margin, no fundamental pattern can be identified. For UC2, a more stable development of F-measure is visible for GA concerning the minimization of O_{sum} considering the confidence interval (cf. Fig. 6c). Maximizing the tolerance margins, on the other hand, leads to deterioration and a large dispersion of the detection performance between experiment runs (cf. Fig. 6d). With minimization of O_{sum}, for UC2, an improvement can be observed with determined configurations inspecting the final F-measure obtained on the training sets ($p = .017$). For UC1, on the test set, the tolerance minimization leads

TABLE IV

DETERMINED OFFSETS FOR THE GRIPPERS' AXES POSITIONS FOR UC1 WITH/WITHOUT HAVING O_{sum} ENABLED AS SECOND OBJECTIVE. EACH CONFIGURATION LEADS TO A COMPLETE AND CONSISTENT DETECTION OF ALL STATES IN THE RESPECTIVE TRAINING SET (F-MEASURE = 1). BP... BASE POSITION, MAP... MAIN ARM POSITION, SAP... SECOND ARM POSITION, WP... WRIST POSITION, GP... GRIPPER POSITION

			Offsets									
	O_{sum}		BP		MAP		SAP		WP		GP	
	min	max	lower	upper	lower	upper	lower	upper	lower	upper	lower	upper
HS	-		.0348	.0756	.0299	.0404	.4878	.0322	.0063	.4977	.0288	.0171
	✓		.0043	.0040	.0282	.0165	.0253	.0529	.0076	.0066	.0343	.0142
		✓	.0208	.2918	.0542	.0208	.6931	.6758	.0085	.6899	.0310	.0247
GA	-		.0092	.0086	.0498	.0096	.0180	.0304	.0077	.0671	.0285	.0407
	✓		.0276	.0113	.0299	.0237	.0075	.0324	.0134	.0014	.0079	.0352
		✓	.0312	.0380	.0388	.0395	.7000	.7000	.0237	.7000	.0184	.2943

to improvements for HS and GA concerning correctness (HS: $p = .001$, GA: $p = 1.364e^{-6}$) and for GA also in completeness ($p = .019$) in detected states. Similarly, minimization leads to better performance on the test sets for UC2, but only for GA, where there are fewer false positives ($p = .001$) and fewer false negatives ($p = .044$).

Concerning the concrete offset values, which induce for each state the tolerance range for each sensor property, these ranges can be set narrow or wider around the designed values if minimizing/maximizing O_{sum} is enabled as a second optimization objective while ensuring complete and consistent detection. For clarification, Table IV shows the tendency in the differences between the properties' lower and upper offset values, albeit all listed configurations provide the same optimal detection performance (F-measure = 1) on the respective training set of randomly sampled execution traces for UC1. For the five investigated axes from the robot (Second Arm Position (SAP), Wrist Position (WP), BP, MAP, GP), different offsets lead to an optimal result. Since there is a dependency between the individual features and their deviations in state detection, not all ranges are reduced or extended to the same extent. Remarkably, it is shown that for individual sensor properties, e.g., `SAP`, even the maximum possible tolerance of .7 works.

6) Interpretation RQ3: The explicit optimization of tolerance levels can lead to improvements in the training phase and the recognition performance on the test sets. Accordingly, the additional minimization is beneficial if the sensor properties differ only very slightly, requiring narrow ranges for differentiation. In particular, when the configurations are used later on the test sets, it becomes apparent that this leads to offsets that better capture the real deviations that are only present to a limited extent in the training set and improvements can be achieved in this respect. Minimization tends to lower tolerance levels in these cases, which benefits in capturing fine differences and avoiding false positives.

Examining the distances between the respective offsets in Table IV, it can be seen that the tolerance level may well be optimized to take into account the longevity of the configuration. With additional maximization, one aims for continued accurate detection despite potentially increasing inaccuracies in the workflow. With minimization, on the other hand, increasing inaccuracies in the workflow can be detected more quickly as they exceed the tolerance ranges, and the detection performance decreases as a result.

D. Discussion

Based on the provided evaluation results and interpretations for the stated research questions, we now critically discuss our approach to characterize further its benefits and limitations as well as possible mitigations.

The presented approach, by using intelligent search, facilitates the reconstruction of realized states from sensor streams to ensure workflow congruence with the logical design model. The evaluation shows that the information in the design models is sufficient for state recognition and, combined with the intelligent search approach, delivers good results. It has to be emphasized that the models are reused without adaptation from the design phase, e.g., already available for synthesizing controllers. This approach streamlines development and facilitates the tracking and subsequent analysis of processes. As a result, it also allows dealing with evolution scenarios where processes need to be adapted, which would only require producing a training set of annotated traces. The rest is automatically derivable from the design models and the usage of the intelligent searchers.

One additional benefit of the presented approach we realized during the evaluation concerns the following point. Individual sensors' relevance and fine adjustment can vary from case to case, even though they may not be immediately recognizable from the design values. Naturally, its tolerance becomes less critical if an axis position remains at the same location throughout the workflow or alternates between two significantly different locations. However, in some cases, a few axis positions or even a single one may be so distinctive in the state sequence that the settings at the remaining axes become negligible. Intelligent searchers, such as GA or HS, can highlight and consider such nuances in the configuration.

Besides these benefits, the following limitations come to light during our evaluation, potentially constraining its applicability and requiring additional research as mitigation.

The first point concerns our filter approach, namely keeping always the first detected instance of a state, sometimes incorrectly returning an instance not contained in the reference set. This can occur when the deviations in the sensor ticks before the instance declared as correct, i.e., the one in the reference set, are too marginal and, thus, still retrieved by the state realization queries. Consequently, the valid instance appears as a duplicate and is discarded. This in itself does not restrict the identification of states according to the workflow, e.g., in a continuous monitoring application. However, the

temporal incorrect assignment can impair analysis purposes, e.g., investigating latencies during state transitions to optimize the process. Further cleaning procedures may be investigated in future research to deal with this issue.

The second point concerns the sample size of the training traces, mainly the range of deviations they cover and on which the configurations are tuned. Hence, our experiments are based on the optimistic assumption that the 10 (randomly selected) executions cover both the finest and the coarsest deviations that will occur in the future at runtime or, in our case, in the 50 remaining test runs. Nevertheless, this assumption lacks practical viability, and additional techniques may be necessary to justify the diversity of systems' execution traces to guarantee proper coverage of different settings that may occur.

Potential mitigations for these identified limitations are as follows. Instead of employing a universal tolerance configuration for the sensor properties, a dedicated configuration may be determined for each state and used in the state realization queries. Axis deviations are therefore addressed separately for each state, which improves query accuracy before filtering and, thus, mitigates the stated problems and increases the computational effort. Our approach's ambition to recognize the states strictly at the recorded times should also be emphasized here. Alternatively, if sufficient for the purpose, this requirement could be relaxed to only check the states' occurrence according to the order in the model. Consequently, the instance delivered by the filter process in the case described no longer leads to rejection of the correct instance and, thus, to a false positive, but an instance that simply reflects the correct realized state, regardless of the reference time of occurrence. This would also make the creation of the reference set optional. The evaluation may be carried out merely based on the number of realized workflow runs in the recorded period. For possible branches in the workflow, however, the alternative of several states would have to be considered. Regarding our evaluation strategy, the split between training and test data inevitably involves the trade-off between finding the most practicable deviations for detection and the confidence in estimating the recognition performance at runtime. However, suppose only the temporal sequence of the recognized states is checked. The reference set may be omitted in such cases, and many more training and test runs can be potentially used. Future studies on the impact on the accuracy when setting permissible tolerances and estimating performance at runtime for such scenarios are of interest.

E. Threats to Validity

In this section, we finally discuss the four types of validity threats [26] that may affect the presented exploratory study.

1) Conclusion Validity: The *conclusion validity* is concerned with the ability to draw correct conclusions about the relationship between treatment and outcome. Furthermore, is it possible to repeat the calculations with the same results? We used two different use cases to counteract this threat, where the respective workflow was executed many times. We ran each search 30 times independently for the results in each use case. This gave us a larger result set and prevented us from drawing conclusions based on outliers. Additionally, we compared the results of HS, GA, and RS with statistical validation.

2) Construct Validity: The *construct validity* is concerned with the extent to which the experimental setting reflects the theory, i.e., is the study well-constructed by using established standards and methods? We used established metrics such as F-measure for our objective function. In this regard, we plan to investigate additional metrics in future work. Since there is hardly any research in the field of logical state detection by meta-heuristics, we compared two different meta-heuristic approaches with each other and with RS to tackle the construct threat. Additionally, we validated our hypotheses by statistical test analysis.

3) External Validity: The *external validity* limits the ability to generalize the results beyond the experiment context. We used two use cases with different settings and complexity in the context of our lab-sized five-axis gripper demonstrator. The scenarios differed in the number of states, structure, and workflow. However, we cannot guarantee that the results are generalizable for all possible production use cases and scenarios. Therefore, further empirical studies in other application domains are necessary to confirm our findings.

4) Internal Validity: The *internal validity* checks if the conducted study is measuring what it is supposed to. This validity threat may affect the independent variables concerning causality, potentially leading to reported results indicating a causal relationship that does not exist. The stochastic nature and the HS and GA parameter settings might be considered an internal validity threat. To address this obstacle, we varied the different control parameters of HS and GA before we performed 30 independent runs for each problem instance.

V. Related Work

With respect to our approach, we discuss research on examining system behavior using runtime data and on tolerance estimation in robotic systems.

A. Examining System Behavior Using Runtime Data

There are several lines of research concerned with providing an understanding of the behavior of a system during operation by examining its runtime behavior based on gathered data at runtime, e.g., *Reverse Engineering* [27] or *Process Mining* [14], [28], [29]. However, only a few studies consider the uncertainty of runtime models, which will be discussed next.

Mayerhofer et al. [30] introduced an approach to capture data uncertainty in software models. For this purpose, they present an approach for capturing aspects of physical entities, such as units, precision, and uncertainties, in software models. In this context, not only "exact" assigned property values have to be expressed in models, but also uncertainty occurs when measuring runtime data. Therefore, extensions of the UML/OCL type `Real`, including operations for computing with uncertain values, are proposed. Vallecillo et al. [31] present a case study where they annotate a "fixed" uncertainty at design time similar to our previous work presented in [8].

They measure and compute different values (including uncertainties) with simulations. In contrast, we consider optimal tolerance ranges to capture uncertainties during operation.

In [28], the authors discover real software behavior from runtime data. For this purpose, they deploy *Process Mining (PM)* techniques [14] to discover the "real" behavior of software components out of unstructured execution traces. Contrary to previous PM algorithms, they consider the hierarchical structure of software, represented by multi-level nested operation calls, by exploiting the call relations among methods. To identify a specific independent run of a single software component, they introduce the notion of a *case*. Such a case determines the scope of the discovered model. In our approach, we focus not only on the discrete behavior by analyzing execution traces but also on the continuous behavior of systems since we employ time series analysis.

Over the years, PM approaches have evolved to deal with complex workflows and noisy data. In [32], the authors propose a genetic PM algorithm for optimization and searching for a suitable fitness function for parsing event traces to generate a process model. They focus on mined models that reflect the behavior of event logs. A challenge the authors are addressing is coping with the lack of negative examples caused by errors in logs. In a follow-up publication [33], the authors employ their GA to deal with noise and incompleteness and perform fine-tuning based on artificial examples. The goal is to find an optimal fit between a process model and an event log. In contrast to our approach, we do not deal with event logs but raw sensor data, which we raise to the level of logical states. Our presented algorithm identifies optimal bounds that can be used to detect logical states correctly. Thus, our produced outcome may be an input for PM algorithms.

Otto et al. [34] present an approach for parameter estimation as input for reusable software components in the automation domain. This approach combines mixed integer nonlinear programming with PM and black-box optimization techniques. The aim is to calculate optimal timing parameter configurations for software components with free parameters for application in discrete manufacturing. By evaluating their approach, the authors calculate the optimal energy consumption based on runtime data. The parameters are adjusted as soon as the method returns a better consumption. In contrast, we compute optimal tolerance ranges to approximate the parameter values of logical states of design models. We use this approximation to enable automatic state detection.

Zou et al. [35] propose an approach to optimize design parameters for automated production systems. They point out that finding efficient control parameters is still common via trial & error. Therefore, they employ simulation optimization techniques to search for optimal control parameters under uncertainty. Our approach does not optimize control parameters but configures tolerance ranges based on execution traces for automatically detecting states in runtime data.

B. Tolerance Estimation for Robotic Systems

Several papers discuss tolerance estimation methodologies for the design of robotic systems to optimize their accuracy and reliability. Zhao et al. [4] present a near-optimal method to estimate joint tolerances for robot arms, e.g., for collision avoidance. They consider safety constraints to provide a lower bound of the maximum permissible deviation from the arm's reference position. In [5], Tipary and Erdős focus on deviations emerging in robotic work cells. Their tolerance modeling approach encompasses various sources of deviations, including workpiece, environment, manipulation, and metrology processes, to assist developers in assessing operation feasibility. Huang et al. [6] propose an optimization model to improve the positioning accuracy reliability of robot manipulators. For this purpose, the latter's kinematic parameters are determined using a custom GA. Their method allows robot designers to evaluate the effects of different parameter tolerances on position accuracy and manufacturing costs. Also, to support runtime monitoring, Narayanan and Bobba [36] propose an anomaly detection framework for robotic arms. At its core, a support vector machine is trained on regular and anomalous workflow executions, whereby for the latter, the workflow is carried out under (task-specific) deviations outside a so-called "tolerance envelope".

The works above mainly focus on aiding in the specification of robotic environments, while our approach aims at runtime behavior observation and analysis for state-based monitoring purposes. Although statistical methods can estimate deviations in sensor data [10], they lack particular support for the state detection task. This includes the significance of individual sensors in distinguishing the workflow states and the potential insignificance of their tolerance ranges in the overall circumstance. In contrast, our search-based method finds an optimal detection rule configuration while considering tendencies toward broader or narrower tolerance ranges. Unlike the approach taken in [36], where measurements are employed for training a model to detect exceptional deviations in a black-box manner, our method employs them to establish a specific configuration for identifying the intended states.

VI. Conclusion and Future Work

In this paper, we presented a novel approach that allows reusing logical design models—as they are available from the engineering phases—in combination with a meta-heuristic approach to tackle the challenge of configuring appropriate tolerance ranges for detecting logical system states based on execution traces collected from sensor value streams. The approach lifts pure sensor data streams to state-based traces of the IT layer and, thus, enables a state-based process view for monitoring and analysis, which can then be further examined with the help of process mining, for example. The evaluation showed that meta-heuristics such as HS or GA are necessary to find meaningful ranges and clearly outperform RS. The degree of differentiation of system states is crucial to whether the identification works well. Thus, if the assigned values of the logical states are too similar, this challenges the search process for optimal tolerance ranges. However, one major lesson learned is that HS can still separate close states better than GA. Furthermore, the dwell time impact could be mitigated by employing a dedicated filter option, although it does not necessarily provide the exact realization

times. Finally, minimizing or maximizing tolerance ranges as an additional objective can support different scenarios that benefit from smaller or higher tolerance range configurations.

To give a brief outlook, we foresee the following steps. First, we plan to expand our study to larger settings by considering more sensor value streams and extensive workflows. Second, we have defined tolerance ranges per property across all logical states. An additional improvement may be achieved if tolerance ranges of system properties were explicitly defined for every state. Finally, based on our findings and produced data sets, additional techniques such as machine learning may be utilized to meet further challenges, e.g., finding the optimal detection time of a state realization.

REFERENCES

[1] Q. Qi et al., "Enabling technologies and tools for digital twin," *J. Manuf. Syst.*, vol. 58, pp. 3–21, Jan. 2021.

[2] T. Wang, J. Cheng, Y. Yang, C. Esposito, H. Snoussi, and F. Tao, "Adaptive optimization method in digital twin conveyor systems via range-inspection control," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 2, pp. 1296–1304, Apr. 2022.

[3] M. Grieves and J. Vickers, "Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems," in *Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches*. Cham, Switzerland: Springer, 2017, pp. 85–113.

[4] W. Zhao, S. He, and C. Liu, "Provably safe tolerance estimation for robot arms via sum-of-squares programming," *IEEE Control Syst. Lett.*, vol. 6, pp. 3439–3444, 2022.

[5] B. Tipary and G. Erdös, "Tolerance analysis for robotic pick-and-place operations," *Int. J. Adv. Manuf. Technol.*, vol. 117, nos. 5–6, pp. 1405–1426, Nov. 2021.

[6] P. Huang, Y. Gu, H. Li, M. Yazdi, and G. Qiu, "An optimal tolerance design approach of robot manipulators for positioning accuracy reliability," *Rel. Eng. Syst. Saf.*, vol. 237, Sep. 2023, Art. no. 109347.

[7] A. Mazak, M. Wimmer, and P. Patsuk-Bösch, "Execution-based model profiling," in *SIMPDA*. Cham, Switzerland: Springer, 2016, pp. 37–52.

[8] S. Wolny, A. Mazak, M. Wimmer, and C. Huemer, *Model-driven Runtime State Identification*. Bogota, Colombia, EMISA, 2019, pp. 29–44.

[9] M. Tlija, B. Louhichi, and A. BenAmara, "Evaluating the effect of tolerances on the functional requirements of assemblies," *Mech. Ind.*, vol. 14, no. 3, pp. 191–206, 2013.

[10] H. Wen, Z. Xiao, A. Markham, and N. Trigoni, "Accuracy estimation for sensor systems," *IEEE Trans. Mobile Comput.*, vol. 14, no. 7, pp. 1330–1343, Jul. 2015.

[11] Z. Woo Geem, J. Hoon Kim, and G. V. Loganathan, "A new heuristic optimization algorithm: Harmony search," *Simulation*, vol. 76, no. 2, pp. 60–68, Feb. 2001.

[12] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications To Biology, Control, and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[13] S. A. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML*. Burlington, MA, USA: Morgan Kaufmann, 2012.

[14] W. Van Der Aalst, "Process mining," *Commun. ACM*, vol. 55, no. 8, pp. 76–83, Aug. 2012, doi: 10.1145/2240236.2240257.

[15] F. Qin, A. M. Zain, and K.-Q. Zhou, "Harmony search algorithm and related variants: A systematic review," *Swarm Evol. Comput.*, vol. 74, Oct. 2022, Art. no. 101126.

[16] X.-S. Yang, "Harmony search as a metaheuristic algorithm," in *Music-Inspired Harmony Search Algorithm*. Cham, Switzerland: Springer, 2009, pp. 1–14.

[17] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, 2nd ed. Cham, Switzerland: Springer, 2015.

[18] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: Past, present, and future," *Multimedia Tools Appl.*, vol. 80, no. 5, pp. 8091–8126, Feb. 2021.

[19] C. C. Aggarwal, *Managing and Mining Sensor Data*. Cham, Switzerland: Springer, 2013.

[20] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2008.

[21] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Softw. Eng.*, vol. 14, no. 2, pp. 131–164, Apr. 2009.

[22] W. N. Venables and B. D. Ripley, *Modern Applied Statistics With S*, 4th ed. Cham, Switzerland: Springer, 2002.

[23] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[24] Z. W. Geem, "Improved harmony search from ensemble of music players," in *Lecture Notes in Computer Science*. Cham, Switzerland: Springer, 2006, pp. 86–93.

[25] M. Ravber, S.-H. Liu, M. Mernik, and M. Črepinsek, "Maximum number of generations as a stopping criterion considered harmful," *Appl. Soft Comput.*, vol. 128, Oct. 2022, Art. no. 109478.

[26] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, and B. Regnell, *Experimentation in Software Engineering*. Cham, Switzerland: Springer, 2012.

[27] L. C. Briand, Y. Labiche, and J. Leduc, "Toward the reverse engineering of UML sequence diagrams for distributed Java software," *IEEE Trans. Softw. Eng.*, vol. 32, no. 9, pp. 642–663, Sep. 2006.

[28] C. Liu, B. van Dongen, N. Assy, and W. M. P. van der Aalst, "Component behavior discovery from software execution data," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2016, pp. 1–8.

[29] V. Rubin, C. W. Günther, W. M. P. Van Der Aalst, E. Kindler, B. F. Van Dongen, and W. Schäfer, "Process mining framework for software processes," in *Proc. Int. Conf. Softw. Process*, 2007, pp. 169–181.

[30] T. Mayerhofer, M. Wimmer, and A. Vallecillo, "Adding uncertainty and units to quantity types in software models," in *Proc. ACM SIGPLAN Int. Conf. Softw. Lang. Eng.*, Oct. 2016, pp. 169–181.

[31] A. Vallecillo, C. Morcillo, and P. Orue, "Expressing measurement uncertainty in software models," in *Proc. 10th Int. Conf. Quality Inf. Commun. Technol. (QUATIC)*, Sep. 2016, pp. 15–24.

[32] A. K. A. de Medeiros, A. J. M. M. Weijters, and W. M. P. van der Aalst, "Genetic process mining: A basic approach and its challenges," *BPM Workshops*, pp. 203–215, 2005.

[33] W. M. P. van der Aalst, A. K. A. de Medeiros, and A. J. M. M. Weijters, "Genetic process mining," in *Proc. ICATPN*, 2005, pp. 48–69.

[34] J. Otto, B. Vogel-Heuser, and O. Niggemann, "Online parameter estimation for cyber-physical production systems based on mixed integer nonlinear programming, process mining and black-box optimization techniques," *At-Automatisierungstechnik*, vol. 66, no. 4, pp. 331–343, Apr. 2018.

[35] M. Zou, F. Ocker, E. Huang, B. Vogel-Heuser, and C.-H. Chen, "Design parameter optimization of automated production systems," in *Proc. IEEE 14th Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2018, pp. 359–364.

[36] V. Narayanan and R. B. Bobba, "Learning based anomaly detection for industrial arm applications," in *Proc. Workshop Cyber-Phys. Syst. Secur. PrivaCy*, 2018, pp. 13–23.

Sabine Sint (Student Member, IEEE) is currently pursuing the Ph.D. degree with the Christian Doppler Laboratory for Model-Integrated Smart Production (CDL-MINT), JKU Linz, with a focus on module reactive model repositories. In addition, she works as a Project Assistant with the Research Unit of Building Physics, TU Wien. Her research interests include model-driven engineering, reverse engineering, and data integration. For more information, please visit https://www.se.jku.at/sabine-sint/.

Alexandra Mazak-Huemer (Member, IEEE) received the Habilitation degree in business informatics. She is working as a PD associated with the Department of Business Informatics - Software Engineering, JKU Linz. She is the Deputy Managing Director of the Research, Science, Innovation and Technology Council. Her research interests include model-driven engineering, information integration, and sustainability in software engineering. For more information, please visit https://se.jku.at/alexandra-mazak-huemer/.

Martin Eisenberg (Student Member, IEEE) received the B.Sc. degree in business informatics and the M.Sc. degree in computer science from JKU Linz. Since joining the CDL-MINT in 2019, he has been involved in research around AI-powered and model-driven technologies. His research interests include data-driven systems, AI applications, and applied machine learning. For more information, please visit https://se.jku.at/martin-eisenberg/.

Daniel Waghubinger is a Student Researcher with the Department of Business Informatics and Software Engineering, JKU Linz. He wrote his bachelor thesis in the research project CDL-MINT. His research interests include web technologies and security. For more information, please visit https://se.jku.at/daniel-waghubinger/.

Manuel Wimmer (Member, IEEE) is a Full Professor with the Department of Business Informatics - Software Engineering, JKU Linz. He is also the Head of the Christian Doppler Laboratory for Model-Integrated Smart Production (CDL-MINT). His research interests include the foundations of model engineering techniques and their application in domains, such as tool interoperability, legacy modeling tool modernization, model versioning and evolution, and industrial engineering. For more information, please visit https://www.se.jku.at/manuel-wimmer/.