Specification Guided Automated Synthesis of Feedback Controllers

NIKHIL KUMAR SINGH, IIT Kanpur, India
INDRANIL SAHA, IIT Kanpur, India

The growing use of complex Cyber-Physical Systems (CPSs) in safety-critical applications has led to the demand for the automatic synthesis of robust feedback controllers that satisfy a given set of formal specifications. Controller synthesis from the high-level specification is an NP-Hard problem. We propose a heuristic-based automated technique that synthesizes feedback controllers guided by Signal Temporal Logic (STL) specifications. Our technique involves rigorous analysis of the traces generated by the closed-loop system, matrix decomposition, and an incremental multi-parameter tuning procedure. In case a controller cannot be found to satisfy all the specifications, we propose a technique for modifying the unsatisfiable specifications so that the controller synthesized for the satisfiable subset of specifications now also satisfies the modified specifications. We demonstrate our technique on eleven controllers used as standard closed-loop control system benchmarks, including complex controllers having multiple independent or nested control loops. Our experimental results establish that the proposed algorithm can automatically solve complex feedback controller synthesis problems within a few minutes.

CCS Concepts: • **Computer systems organization** → **Embedded systems, Cyber Physical Systems**; • **Computing Methodology** → *Modelling and Simulation, Matrix Decomposition, Multi-parameter Optimization*; • **Software and its engineering** → Formal Methods.

Additional Key Words and Phrases: Feedback control, controller synthesis, multi-parameter tuning, parametric signal temporal logic, falsification

ACM Reference Format:
Nikhil Kumar Singh and Indranil Saha. 2021. Specification Guided Automated Synthesis of Feedback Controllers. In *EMSOFT '21: EMSOFT '21: ACM SIGBED International Conference on Embedded Software (EMSOFT), October 10–15, 2021, Virtual Conference.* ACM, New York, NY, USA, 25 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Modern cyber-physical systems (CPSs) are used in many safety-critical applications such as transportation, agriculture, medical devices, smart grids, space explorations, to name a few. Most cyber-physical systems involve sophisticated feedback controllers for regulating the behavior of the physical processes. The overall efficiency and correctness in the CPS behavior depend significantly on the performance of the feedback controllers.

The CPSs used in safety-critical applications need stringent correctness guarantees on their behavior. Formal verification techniques based on translating a closed-loop system to hybrid

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Embedded Software (EMSOFT), 2021.
Authors' Address: N. K. Singh and I. Saha. Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, Uttar Pradesh 208016, India. Email: {nksingh,isaha}@cse.iitk.ac.in.

EMSOFT '21, October 10–15, 2021, Virtual Conference

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

automata and then performing reachability analysis can provide guarantees on the behavior of the closed-loop system with respect to safety properties [7]. However, there are different classes of properties, often involving timing constraints, that are beyond the scope of this approach. For such specifications, runtime monitoring and falsification based methodologies [5] are employed to gain confidence in the feedback controller. A major deficiency of the formal and semi-formal verification methodologies is that they provide almost no guidelines on how to improve the feedback controllers so that the verification of the closed-loop system succeeds with respect to the formal specifications.

An alternative to formal verification for ensuring the reliability of safety-critical systems is the automated synthesis of the system from a set of formal specifications. In the context of CPS, the synthesis problem is the following: Given a physical process and a set of formal specifications, could we synthesize a feedback controller so that the closed-loop system would satisfy all the specifications? Synthesis of feedback controllers has been widely studied in control theory, where the form of the controller is decided first, and then the parameters of the controller are tuned based on some experimental procedure to ensure that the controller provides satisfactory closed-loop behavior. For example, the widely used PID controller has parameters corresponding to the proportional, integral, and derivative gains, which can be tuned using the well-established Ziegler-Nichols method [38] or other tuning methodologies [4, 52]. Although MATLAB provides a PID tuner that lets us choose optimal PID values, this lacks on the following counts. First, it only uses a qualitative notion of performance and robustness for tuning, while specifications in CPSs are versatile and complex. Second, it still requires manual intervention to provide a balance between the two metrics.

Of late, there have been significant efforts towards formalizing specifications of the closed-loop control system as signal temporal logic (STL) formulas [25]. This enables us to use STL formulas as formal specifications for synthesizing feedback controllers for CPSs. While feedback controller synthesis is a well-researched area, to the best of our knowledge, synthesizing them to satisfy a set of signal temporal logic specifications has not been addressed in the literature yet.

In this paper, we propose a technique for controller synthesis for complex nonlinear systems using specifications given in STL. This technique is based on rigorous analysis of the controller parameters involved in the generation of control inputs. Often the controller synthesis involves tuning of multiple parameters. We tune different parameters over successive iterations by selecting which parameter has the most impact on the specification violation at that moment. Hence, we provide an incremental and iterative parameter tuning based controller synthesis algorithm. Moreover, it is worth mentioning that this technique is not limited to any particular class of controllers (as in the case of the MATLAB PID tuner) but is applicable to all parameterized controllers. In case a feedback controller that satisfies all the specifications cannot be synthesized, i.e., our algorithm stops without finding a solution, we identify the maximal subset of the specifications that can be satisfied by a controller. Then, we adjust the parameters of the unsatisfied specifications so that the closed-loop system with the synthesized controller satisfies them now.

We demonstrate the effectiveness of our method by synthesizing controllers (even complex ones having multiple nested control loops) that satisfy a set of complex STL specifications. Our experimental results establish that our proposed technique can synthesize feedback controllers for complex nonlinear systems within a few minutes.

Our contribution can be summarised as follows:

- We present an algorithm for synthesizing multi-parameter complex feedback controllers for cyber-physical systems to enable them to satisfy a given set of signal temporal logic specifications. To the best of our knowledge, this is the first work that synthesizes parameter-based feedback controllers that satisfy a set of complex STL specifications.

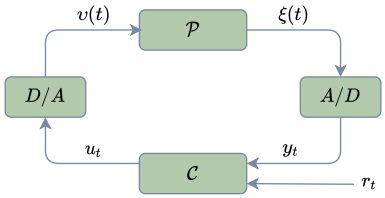


Fig. 1. A closed-loop control system

- In case our algorithm cannot synthesize such a controller, we find a maximal subset of the specifications that can be satisfied by a synthesized controller. Then, we tune the parameters of the unsatisfied specifications so that the closed-loop system with the synthesized controller satisfies all specifications. Thus, we provide a complete controller design solution involving both controller parameter tuning and specification mining.
- We implement our algorithm in a MATLAB based automated tool. We apply our software to synthesize the feedback controller in the form of a parameterized controller for eleven systems, including complex nested/cascaded/multi-loop controllers.

2 PROBLEM

2.1 Preliminaries

2.1.1 Closed-loop Control System with Parameterized Controller. In a closed-loop control system (CCS), the output of the system is used to generate control input that is applied to the system to regulate its behavior. Open-loop systems often either are not stable or do not satisfy desired properties. Control engineers design feedback controllers for open-loop systems so that the closed-loop system becomes stable and also satisfies many desired properties. In Figure 1, a block diagram of a closed-loop system is shown, where the plant $\mathcal{P}$ with state vector $\chi(t)$ generates the output $\xi(t)$ according to the following dynamical equations:

$$\dot{\chi}(t) = f(\chi(t), v(t)), \tag{1}$$

$$\xi(t) = g(\chi(t)). \tag{2}$$

Here $v(t)$ denotes the continuous-time control signal applied to the plant. To generate the control signal, the output of the plant is fed to an A/D converter. The output of the A/D converter is the discrete-time output signal y_t, which is fed to the controller C along with a discrete-time reference signal with value r_t at time t. The discrete-time control signal u_t is processed through a D/A converter that produces the continuous-time control signal $v(t)$.

The feedback controller C may contain multiple parameters denoted as vector P. The control generated by this controller is given by

$$u_t = k_{\mathsf{P}}(y_t, r_t). \tag{3}$$

Here, k_{P} is the controller function with a parameter vector P whose size is equal to the number of parameters in the controller. For example, in the case of a PID controller k_{P} would be a vector of three elements - the proportional, integral, and derivative gains [52].

The model of a CCS ($\mathcal{M}$) is defined as the composition of the controller and the plant, i.e.,

$$\mathcal{M} = \mathcal{P} \circ C, \tag{4}$$

where $\circ$ denotes the synchronous side-by-side composition with feedback [16]. We denote the state of the system $\mathcal{M}$ at time t by $z_t \in \mathcal{Z}$, where $z_t = (r_t, y_t)$ and $\mathcal{Z}$ denotes the statespace of the system. A trace $\omega = (z_0, z_1, \ldots)$ is defined as the sequence of states of the system evolving with discrete time-steps. We use $\mathcal{L}(\mathcal{M})$ to denote the set of all traces of $\mathcal{M}$.

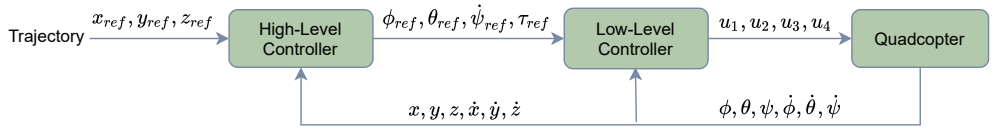


Fig. 2. A model of PX4 AutoPilot [37] for a quadcopter, the low-level controller here runs in a cascaded loop

A closed-loop control system can be highly complex, representing very powerful CPSs. The controllers for such systems may have multiple simpler controllers nested or cascaded to generate a more complex control. For instance, Figure 2 shows the block diagram of standard PX4 autopilot for a quadcopter [37], where there are two controllers in a cascaded loop - one controls the position and the other controls the orientation. Tuning the parameters of such controllers is a challenging task. We have used several such controllers in our experiments (refer to Section 4).

2.1.2 Signal Temporal Logic. Signal Temporal Logic (STL) [30] is an extension over Metric Temporal Logic (MTL) [27] and Linear Temporal Logic [41]. It consists of real-time predicates over the signals. It provides us the capability to reason about the real-time properties of signals (simulation traces).

The syntax of an STL specification ϕ is defined by the grammar

$$\phi = \text{true} \mid \pi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \, U_I \, \phi_2, \tag{5}$$

where $\pi \in \Pi$, Π is a set of atomic predicates, and $I \subseteq \mathbb{R}^+$ is an arbitrary interval of non-negative real numbers. The operators $\neg$ and $\wedge$ denote logical NOT and AND operators. Other logical operators like logical OR ($\vee$) or implication ($\implies$) can be derived using $\neg$ and $\wedge$. The temporal operator U_I (until) implies that ϕ_2 becomes true within the time interval I and ϕ_1 must remain true until ϕ_2 becomes true. Two other popular operators are eventually ($\Diamond_I$) and always ($\Box_I$), which can be derived from the operators defined above. The formula $\Diamond_I \, \phi$ means that the formula ϕ will be true sometime in the time interval I. The formula $\Box_I \, \phi$ means that the formula ϕ will always be true in the time interval I. We use the temporal operators U, $\Diamond$ and $\Box$ to denote the operators U_I, $\Diamond_I$ and $\Box_I$ with the time interval I to be $[0, \infty)$.

Robustness Semantics of STL: The robustness semantics of STL used in this paper is defined in [15]. We use Euclidean metric as the norm to measure the distance d between two values $v, v' \in \mathbb{R}$, i.e., $d(v, v') = \| \, v - v' \, \|$. Let $v \in \mathbb{R}$ be a value, $A \subseteq \mathbb{R}$ be a set. Then the *signed distance* from v to A is defined as:

$$\texttt{Dist}(v, A) = \begin{cases} inf\{d(v, v') \mid v' \notin A\} & \text{if } v \in A, \\ -inf\{d(v, v') \mid v' \in A\} & \text{if } v \notin A. \end{cases} \tag{6}$$

Intuitively, $\texttt{Dist}(v, A)$ measures how far a value v is from the violation of the inclusion in the set A. In both cases, we search for the minimum distance between v and a point on the boundary of A. As2 the case $v \notin A$ refers to a violation, and thus the negative sign is used in the definition.

We use $\mathrm{O} : \Pi \to 2^{\mathcal{Z}}$ to denote the mapping of a predicate π to a set of states. Given a trace $\omega = (z_0, z_1, \ldots, z_t, \ldots)$ and the mapping O, we define the robust semantics of ω w.r.t. ϕ at time $t \in \mathbb{R}$, denoted by $[[\phi]](\omega, t)$, by induction as follows:

$$[[\text{true}]](\omega, t) = +\infty, \tag{7a}$$

$$[[\pi]](\omega, t) = \texttt{Dist}(z_t, \mathrm{O}(\pi)), \tag{7b}$$

$$[[\neg\phi]](\omega, t) = -[[\phi]](\omega, t), \tag{7c}$$

$$[[\phi \wedge \psi]](\omega, t) = \texttt{min}([[\phi]](\omega, t), [[\psi]](\omega, t)), \tag{7d}$$

$$[[\phi \, U_I \, \psi]](\omega, t) = \sup_{t' \in t+I} \texttt{min}([[\psi]](\omega, t'), \inf_{t'' \in [t, t']} [[\phi]](\omega, t'')). \tag{7e}$$

If $[[\phi]](\omega, t) \neq 0$, its sign indicates the satisfaction status. Also, if ω satisfies ϕ at time t, any other trace ω' whose Euclidean distance from ω at time t is smaller than $[[\phi]](\omega, t)$ also satisfies ϕ at time t. The robustness metric $[[\phi]]$ maps each simulation trace ω to a real number ρ. Intuitively, the robustness of a trace $\omega \in \mathcal{L}(\mathcal{M})$ with respect to an STL formula ϕ is the radius of the largest ball centered at trace ω that we can fit within $\mathcal{L}_\phi$, where $\mathcal{L}_\phi$ is the set of all signals that satisfy ϕ.

Falsification: We define the falsification problem as follows: For a given system $\mathcal{M}$, a specification ϕ, and a simulation time T, find $\bar{\omega}= \langle z_0, z_1, \ldots, z_T \rangle$ such that $\omega = (z_0, z_1, \ldots, z_T, \ldots) \in \mathcal{L}(\mathcal{M})$ and $[[\phi]](\omega, t) < 0$. This is generally captured as an optimization problem:

$$\omega^* = \underset{\omega \in \mathcal{L}(\mathcal{M})}{\arg\min} \; [[\phi]](\bar{\omega}), \tag{8}$$

where, we define $[[\phi]](\bar{\omega})$ as the minimum robustness of trace $\bar{\omega}$ w.r.t. ϕ.

2.1.3 Parameteric Signal Temporal Logic. A Parameteric Signal Temporal Logic (PSTL) [2] formula is an STL formula template where the numeric constants are represented by symbolic parameters. Let us denote the parameters as $p = (p_1, ...p_n)$, where $p \in \mathbb{R}^n$. Let π denotes numerical predicates of the form $f(z_t) < c$. The syntax of a PSTL specification ϕ is defined as:

$$\phi_p = \pi \mid \neg\phi_p \mid \phi_p \wedge \phi_p \mid \phi_p \, U_{[\tau_1, \tau_2]} \, \phi_p. \tag{9}$$

By substituting the value $v(p) \in \mathbb{R}^n$ for the parameters p in a PSTL formula ϕ_p, we get an STL formula $\phi_p(p = v(p))$. Different values of these parameters give us different signals that satisfy the PSTL specification.

The valuation $v(p)$ is called a *δ-tight* valuation, if a small perturbation in $v(p)$ makes the PSTL specification unsatisfiable w.r.t. $\mathcal{M}$ i.e $\mathcal{M} \nvDash \phi_p(p = v(p) + \delta)$ or $\mathcal{M} \nvDash \phi_p(p = v(p) - \delta)$. This is important because valuations of the too relaxed parameters correspond to STL specifications with less restrictive constraints.

The *polarity* $\zeta(p, \phi)$ of a parameter p w.r.t. a PSTL formula ϕ is positive if increasing the value of p increases the robustness of satisfaction of ϕ and negative otherwise. A formula ϕ is *monotonic* w.r.t given parameters if all the parameters are of fixed polarity [2].

2.1.4 Standard specifications for a CCS. A closed-loop control system has several desired specifications, such as *settling time*, *rise time*, *convergence*, *maximum overshoot*, and *smoothness*. The goal of synthesizing a feedback controller k_P for a plant $\mathcal{P}$ is to find the values of the parameters P in such a way that the closed-loop system $\mathcal{M}$ satisfies all the specifications. Recently, Kapinsky et al. [25] have provided the templates of the specifications in STL. We use them as the formal specifications while synthesizing a feedback controller for a plant. The template STL specifications are presented below. Each specification consists of predicates on output y_t produced by plant $\mathcal{P}$ under the control of C. The reference signal is denoted by r_t. We consider *step signal* as the reference input pattern in defining the template STL specification as step-response is routinely used to characterize the behaviour of the closed-loop control system. We assume that step takes place at $t = 0$ (step-time).

- **Settling time.** It is defined as the time taken by the output of the CCS to enter and remain within a specific error band (ϵ_1). Formally, $\Box_{[\tau_s, T]}(|y_{t+\delta_1} - y_t| < \epsilon_1)$, where, τ_s is the settling time, T refers to the time when the simulation ends and δ_1 is a small time offset.
- **Rise time.** It is defined as the time required (τ_r) for the output signal y_t to reach β-times ($0 < \beta \leq 1$) of its final value from its initial value, i.e., $\Diamond_{[0, \tau_r]}(y_t > \beta \cdot r_t)$.
- **Convergence.** This specification ensures that the actual output y_t converges with the desired output r_t within a specified time (τ_c) and range (ϵ_2), i.e., $\Diamond_{[0, \tau_c]}\Box(|(y_t - r_t| < \epsilon_2)$.
- **Overshoot.** This specification enforces a bound on the output exceeding its final, steady-state value. Formally, $\Box(y_t < \alpha \cdot r_t)$, where α enforces the bound.

- **Smoothness** This condition ensures that there are no sharp changes (spikes) in the output signal values. This is because sharp changes can lead to unacceptable system performance. This sharp change is specified by slope μ (w.r.t time-shift δ_2) in the following specification. Formally, $\square(\neg((up) \wedge \Diamond_{[0,\tau_{sm}]}(down)))$, where, $up := \frac{(y_{t+\delta_2}-y_t)}{\delta_2} > \mu$, $down := \frac{(y_{t+\delta_2}-y_t)}{\delta_2} < -\mu$. Here, τ_{sm} is the width of the spike.

The reason of using step-time $t = 0$ for our step-response input is simplification. Consider the case of overshoot specification written with a generalised step-response of form "$\square_{[0,T]}(\texttt{step}(r_t) \implies \square(y_t < \alpha \cdot r_t)$". Now, the time for which the predicate $\texttt{step}(r_t)$ is `false`, the whole specification remains `true`. This specification is violated only when predicate $\texttt{step}(r_t)$ is `true` and predicate $\square(y_t < \alpha \cdot r_t)$ is `false`. Thus, in our simplification, we use step-time $t = 0$ and check the predicate $\square(y_t < \alpha \cdot r_t)$ for violation instead.

2.2 Problem Definition

The controller C is associated with a function k_{P}, where P denotes the tuple containing the controller parameters. We denote by $\mathsf{P}.\kappa$ the κ-th parameter of the controller. The controller C with the values *pval* for the parameters P is denoted by $C[\mathsf{P} \leftarrow pval]$.

An *atomic specification* ϕ represents an acceptable behaviour of a CCS $\mathcal{M}$ (for example, settling time). The *specification* Φ for a CCS is given as the conjunction of several atomic specifications $\phi_1, \phi_2, \ldots, \phi_m$, i.e., $\Phi \equiv \phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_m$. In some context, we also represent the specification Φ as the set of all its atomic specifications, i.e., $\Phi = \{\phi_1, \phi_2, \ldots, \phi_m\}$. Any subset $\{\phi'_1, \phi'_2, \ldots, \phi'_l\}$ of Φ is called a *subspecification* of Φ. Thus, for $\{\phi'_1, \phi'_2, \ldots, \phi'_l\} \subseteq \Phi$, $\Phi' = \phi'_1 \wedge \phi'_2 \wedge \ldots \wedge \phi'_l$ is a subspecification of Φ. Any atomic specification of Φ is a subspecification of Φ.

For an atomic specification ϕ to get satisfied by $\mathcal{M}$, any trace $\omega \in \mathcal{L}(\mathcal{M})$ should belong to the language of ϕ, i.e., $\forall \omega \in \mathcal{L}(\mathcal{M}),\ \omega \in \mathcal{L}_\phi$. However, for an atomic specification ϕ, if there exists a trace $\omega' \in \mathcal{L}(\mathcal{M})$ such that ω' does not belong to the language of ϕ, i.e., $\exists \omega' \in \mathcal{L}(\mathcal{M}),\ \omega' \notin \mathcal{L}_\phi$, then the model $\mathcal{M}$ does not satisfy the atomic specification ϕ, and we write $\mathcal{M} \not\models \phi$. In such a situation, our goal is to tune the controller parameters such that the system satisfies the specification. Ideally, after the tuning process, we would like $\mathcal{M}$ to satisfy all the atomic specifications in Φ. However, after sufficient tuning of the controller parameters, if we fail to get $\mathcal{M}$ to satisfy an atomic specification ϕ', then that may be due to the reason that ϕ' itself is not consistent with the other atomic specifications in Φ. In that case, we attempt to tune the unsatisfied atomic specification ϕ' in such a way that $\mathcal{M}$ with the tuned controller satisfying the other atomic specifications now also satisfy modified ϕ'.

The problems addressed in this paper are formally presented below.

Problem 1. For a given plant $\mathcal{P}$, a template controller C with the set of parameters P, and a specification Φ, find the values *pval* for the controller parameters in P such that the CCS $\mathcal{M}$ obtained by composing $\mathcal{P}$ and $C[\mathsf{P} \leftarrow pval]$ satisfies Φ, i.e., mathematically, $\mathcal{P} \circ C[\mathsf{P} \leftarrow pval] \models \Phi$. If there does not exist a valuation *pval* for P such that $\mathcal{P} \circ C[\mathsf{P} \leftarrow pval] \models \Phi$, find the maximal subspecification set Φ_{maxsat} of Φ ($\Phi_{maxsat} \in 2^\Phi$) and the values *pval* for the controller parameters in P such that $\mathcal{P} \circ C[\mathsf{P} \leftarrow pval] \models \Phi_{maxsat}$.

Problem 2. For the unsatisfiable subspecification Φ_{unsat}, given by $\Phi \setminus \Phi_{maxsat}$, find the revised subspecification Φ_r such that $\mathcal{P} \circ C[\mathsf{P} \leftarrow pval] \models \Phi_{maxsat} \wedge \Phi_r$.

2.3 Example

We illustrate the problem with an example of a Quadcopter-SISO (single-input single-output) CCS, shown in Figure 3. This CCS $\mathcal{M}$ contains a PID controller [52] (light gray) which controls the

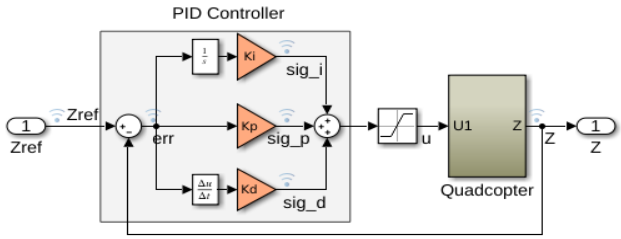


Fig. 3. Quadcopter-SISO model. The area in light gray represents the controller. The orange blocks represent the tunable parameters. The Quadcopter plant is represented in dark gray.

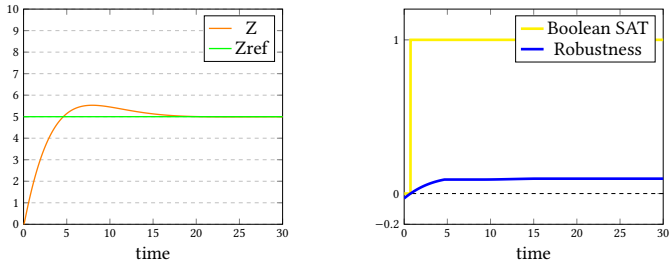


Fig. 4. Falsification for Quadcopter-SISO model. In the figures, we plot the signals $Zref$, Z and the satisfaction (boolean and quantitative) for ϕ.

Quadcopter. The terms K_p, K_i, and K_d refer to the gain constants associated with the proportional, integral, and derivative terms.

The input to the controller is the reference Z-coordinate $Zref_t$. The Quadrotor is allowed to move only along the Z-axis, and hence Z_t is the actual output. The PID controller generates the control u_t using the error signal $e_t = Zref_t - Z_t$. The input $Zref_t$ can take any value between $[1, 5]$ during the simulation. The default values of K_p, K_i and K_d are 5, 1, and 10, respectively.

Let us consider the following specification: The actual output Z_t converges to the desired output $Zref_t$ (with error band ϵ) within τ seconds. We express this as the following STL specification:

$$\phi = \Diamond_{[0,\tau]} \Box (|Z_t - Zref_t| < \epsilon). \tag{10}$$

When we simulate the model of $\mathcal{M}$ w.r.t. ϕ with $\tau = 15$ and $\epsilon = 0.1$ for 30s, the specification is falsified. The timestamps where specification violation occurs are those where the robustness value is negative (refer to Figure 4). Here, we assume that the plant model is correct, and we want to generate correct control signals for this plant such that the specification is satisfied. Thus, the goal is to find the correct values for the controller parameters K_p, K_i, and K_d. As we will see later, the correct controller parameter values found using our algorithm that satisfies ϕ is (7.5, 1, 10). Here, the increase in proportional gain makes the system more responsive to error and hence helps achieve convergence faster.

We can monitor the simulation traces of the system for the specifications ϕ using various tools [1, 14, 40]. In case of violation of a specification ϕ, the information provided by these tools is not sufficient for fixing the controller. So, we need an automated procedure that can help us tune the controller parameters of $\mathcal{M}$. In the next section, we propose an algorithm for *controller synthesis* (Problem 1). We will also show how our basic algorithms can be extended to solve the maximal satisfiable subspecification identification and subspecification repair (Problem 2) problems. We will illustrate our algorithms using this example.

3 ALGORITHM

3.1 Controller Synthesis

3.1.1 NP-Hardness. The standard SOF (Static Output Feedback) stabilization problem in control theory is a well-known NP-Hard problem [6]. This complexity result holds even for simple linear systems with a gain parameter in the controller while considering the stability property. On the other hand, in our controller synthesis problem, (a) we consider general nonlinear systems, (b) our controller parameters are more general, and (c) the STL specifications that we consider are more general than the stability property. Specifically, the convergence specification is stronger than stability, as it requires the output signal to converge within a given time-bound. Hence, the SOF problem can be reduced to our controller synthesis problem in polynomial time. This implies that our controller synthesis problem is as hard as the original SOF problem. The above discussion leads to the following theorem.

Theorem 3.1. *[NP-Hardness] The controller synthesis problem (Problem 1) described in Section 2.2 is NP-Hard.*

The above complexity result encourages us to explore heuristics based solutions like multi-parameter tuning to solve the controller synthesis problem.

3.1.2 Controller Synthesis Algorithm. In this section, we present our controller synthesis algorithm (Algorithm 1). The algorithm takes as input a plant $\mathcal{P}$, a parameterized controller C, and an STL specification Φ. Note that Φ can have two different meanings depending on the context - it can represent an STL formula as a conjunction or as a set of the atomic specifications.

In our algorithm, we iteratively synthesize a new controller that, after each iteration, is incrementally better than the previous controller in terms of robustness of the specifications. A parameterized controller consists of multiple tunable parameters. In each iteration, we identify the parameter whose tuning has the potential to result in the maximum improvement in robustness (w.r.t. specification). For this purpose, we use a matrix decomposition technique where we find the most significant parameter corresponding to the largest singular value of the matrix that contains the falsified portion of the traces associated with the controller parameters. As the chosen parameter has the maximum contribution towards the falsification of the given specification, we focus on tuning it in the current iteration.

We now describe the algorithm in detail. In line 2, we compose the plant $\mathcal{P}$ and the controller C to generate the CCS $\mathcal{M}$. In line 3, ρ_{cur} stores the current minimum robustness of the falsification of the model $\mathcal{M}$ w.r.t. Φ. In falsification, we uniquely identify the most critical point in the falsified trace using the minimum robustness value. Most falsifiers return this value whenever a specification is violated. In line 4, we store the default values of all the parameters of C in $default\_val$. In line 6, we initialize two parameters - ∂_l and ∂_r - in order to explore the values smaller and larger than $default\_val$. The new_val variable stores the default values of controller parameters (line 7). In lines 9-46, we run the main loop until we are able to generate new values for ∂. In our implementation, we start with a default value of $\eta = 0.5$ and update it (line 45) when necessary conditions are satisfied.

In line 10, the `select_parameter` procedure gives us the index κ of the parameter which is the largest contributor to the falsification of specification Φ. In lines 11-20, we find the minimum robustness for two controllers (i.e. ρ_l and ρ_r) - one with corresponding parameter larger than $new\_val[\kappa]$, i.e., $new\_val[\kappa] * \partial_r$, and the other with corresponding parameter smaller than $new\_val[\kappa]$, i.e., $new\_val[\kappa] * \partial_l$. Note that we simply check robustness for the controllers at the given values, but we do not assign these values to the controller parameters. Such assignments only happen at line 27 and 37. Then, we check if either of those robustness values is positive, i.e., the specification Φ is

Algorithm 1: Controller Synthesis Algorithm

```
1  procedure controller_synthesis (P, C, Φ)
2      M ← P ∘ C
3      ρ_cur ← min_robustness(M, Φ)
4      default_val ← get_parameter_values(C)
5      max_count ← 0 ; Φ_mxl ← ∅
6      ∂_l ← 1 − η; ∂_r ← 1 + η
7      new_val ← default_val
8      /* starting exploration for controller parameter values */
9      while |∂_l − ∂_r| > tol_∂ do
10         κ ← select_parameter(P,C, Φ)
11         M_l ← P ∘ C[P.κ ← new_val[κ] ∗ ∂_l]
12         ρ_l ← min_robustness(M_l, Φ)
13         if ρ_l > 0 then
14             C ← C[P.κ ← new_val[κ] ∗ ∂_l]
15             return C
16         M_r ← P ∘ C[P.κ ← new_val[κ] ∗ ∂_r]
17         ρ_r ← min_robustness(M_r, Φ)
18         if ρ_r > 0 then
19             C ← C[P.κ ← new_val[κ] ∗ ∂_r]
20             return C
21         ctr ← 0
22         if ρ_r > ρ_l then
23             /* exploring larger parameter values */
24             while (|ρ_r − ρ_cur| > tol_ρ) ∧ (ctr < H) do
25                 ρ_cur ← ρ_r
26                 new_val[κ] ← new_val[κ] ∗ ∂_r
27                 C ← C[P.κ ← new_val[κ]]
28                 M ← P ∘ C
29                 ρ_r ← min_robustness(M, Φ)
30                 if ρ_r > 0 then return C
31                 ctr ← ctr + 1
32         else
33             /* exploring smaller parameter values */
34             while (|ρ_l − ρ_cur| > tol_ρ) ∧ (ctr < H) do
35                 ρ_cur ← ρ_l
36                 new_val[κ] ← new_val[κ] ∗ ∂_l
37                 C ← C[P.κ ← new_val[κ]]
38                 M ← P ∘ C
39                 ρ_l ← min_robustness(M, Φ)
40                 if ρ_l > 0 then return C
41                 ctr ← ctr + 1
42         ⟨Φ_mxl, max_count⟩ ← maximal_sat_spec (M,
             Φ,Φ_mxl, max_count)
43         /* reset if exploration goes too far */
44         if ‖ new_val − default_val ‖> tol then
45             ⟨∂_l, ∂_r⟩ ← update(∂_l, ∂_r)
46             new_val ← default_val
47     return C

48 /* Find the index κ of the parameter which is the largest
       contributor to the falsification of specification Φ */
49 procedure select_parameter (P, C, Φ)
50     M ← P ∘ C
51     ω_f ← falsify(M, Φ)
52     if ω_f = ∅ then return ∅
53     γ ← controller_parameters_signals(C)
54     /* consolidating system traces into matrix X */
55     X ← [ω_f.τ]
56     for i = 1 : length (γ) do
57         s ← plot_sig_portrait (ω_f, γ[i])
58         X ← [X s]
59     /* getting the robustness plot for the falsified trace */
60     ρ ← plot_robust_sat (ω_f, Φ)
61     /* consolidating timestamps where specification is
         violated into matrix Y */
62     Y ← [ ]; j ← 1
63     for i = 1 : length (ρ) do
64         if ρ[i] < 0 then
65             Y[j] ← [τ[i] ρ[i]]; j ← j + 1
66     /* consolidating system traces where specification is
         violated into matrix W */
67     W ← join(X, Y, τ)
68     W ← remove_columns (W, {τ, ρ})
69     /* finding the parameter corresponding to the largest
         singular value */
70     [Q, R, E] ← matrix_decomposition(W)
71     κ ← E[1]
72     return κ

73 /* Find the maximal subset of Φ which is satisfied by the
       current closed loop system, and add it to the set Φ_mxl */
74 procedure maximal_sat_spec (M, Φ, Φ_mxl, max_count)
75     count ← 0 ; Φ_sol ← ∅
76     for φ ∈ Φ do
77         if min_robustness(M, φ) > ϵ then
78             Φ_sol ← Φ_sol ∪ {φ}
79             count ← count + 1
80     /* if more sub-specifications are satisfied than the
         current count */
81     if count > max_count then
82         Φ_mxl ← {Φ_sol}
83         max_count ← count
84     else
85         /* if another combination of sub-specs is satisfied */
86         if count = max_count ∧ Φ_sol ∉ Φ_mxl then
87             Φ_mxl ← Φ_mxl ∪ {Φ_sol}
88     return ⟨Φ_mxl, max_count⟩
```

satisfied (line 13 and 18). In line 21, we initialize a counter ctr, which is used to ensure that we never exceed our computation budget. Now, we compare the two robustness values and move in the direction where robustness improves (lines 22 -41). In the while loops (lines 24-31 and lines 34-41), we keep on moving in the same direction until either of these two conditions satisfy - (1) the robustness stops improving and (2) we have exhausted our computation budget (denoted by H). In line 27 and line 37, we assign $new\_val[\kappa]$ to the κ-th controller parameter $\mathtt{P}.\kappa$. In line 42, the `maximal_sat_spec` function tracks the maximum number of sub-specifications (max_count) satisfied so far and also the corresponding maximal set Φ_{mxl}. These two variables are initialised in line 5. In lines 44-46, we ensure that our exploration for new parameter values is not too far from the default parameter values (in terms of Euclidean distance). In case we deviate from the specified range, we update the values of ∂_l and ∂_r. Intuitively, by updating those values, we control the *speed* of this exploration of parameter values (line 45). Here, we assume that the correct parameter values that fix the model lie within a certain distance from the default parameter values. This is a reasonable assumption since we assume that the default values are set using the insight of an experienced control engineer. In this way, we synthesize the controller incrementally.

An important point worth mentioning here is that our exploration strategy for new parameter values is not static but dynamic, i.e., we do not control the *direction* of exploration but always move in the direction in which the robustness improves. However, we do control the *speed* of this exploration, as discussed above.

In lines 49- 72, we define the function `select_parameter`. In line 51, we find the trace ω_f of the model $\mathcal{M}$ that causes the falsification of Φ. In line 53, γ represents all the signals associated with the controller parameters. For example, in the model shown in Figure 3, the signals are *sig_p*, *sig_i*, and *sig_d*. In line 55, we create a matrix `X` with one column containing all the time-stamps τ of the trace ω_f. Mathematically, $\tau = [0\ h\ 2h \ldots Th]^{\mathsf{T}}$, where h is the duration of a simulation step and T is the length of the trace ω_f, i.e. $T = |\omega_f|$. In lines 56-58, we add values to matrix `X` with the columns representing the traces for the signals in γ for all time-stamps. In line 60, we plot the robustness (ρ) of ω_f with respect to Φ. In lines 63-65, we create a matrix `Y` containing the timestamps and the robustness values where Φ is violated, i.e., the robustness values are negative. Although an STL specification is satisfied by the trace only at the beginning, i.e., at $t = 0$, the falsification of an STL property by a trace is indicated by the negative robustness of the specification on the trace at any time point. The plot in line 57 differs from the plot in line 60. While the earlier is a simple portrait of values of a signal, the latter is the robust satisfaction of a signal.

In line 67, we join matrices `X` and `Y` with column τ using the `INNER JOIN` procedure [48] and remove the columns τ and ρ, and store it in matrix `W` (line 68). The matrix `W` contains values of the controller parameter signals γ that correspond to negative robustness (falsification). In lines 70-71, we find the most significant component of `W` using `matrix_decomposition` function which is described later. This means that we get the index of controller parameter κ that has the largest contribution toward the falsification of Φ.

In lines 74-88, we define the function `maximal_sat_spec`. In lines 76-79, we check for the maximum number and corresponding solution set (Φ_{sol}) of sub-specifications that is satisfied for the current synthesized controller. If the count is greater than the maximum count found so far, we make $\{\Phi_{sol}\}$ as our maximal set (lines 81-83). If we find another solution set for the same maximal count, we append it to the maximal set (lines 86-87).

Note that our algorithm can account for noises and disturbances by introducing them as additional inputs to the system.

3.1.3 Matrix decomposition for selecting the right controller parameter. Here we analyze the matrix `W` that contains the values of signals corresponding to the controller parameters, where robustness

becomes negative. We need to identify the column in matrix W (corresponding to a controller parameter) that has the maximum information about the matrix W, as the corresponding parameter has the potential to have the maximum impact on the generation of the matrix. This column corresponds to the largest singular value of W. The concept of *singular values* [19] of the matrix is applicable for the rectangular matrices, which can be related to the eigenvalues of a square matrix. The *singular values* of any matrix A is the square-root of the eigenvalues of their associated square Gram matrix (A^TA) [19]. The rank of a rectangular matrix is given by the number of its non-zero singular values.

To compute the singular values of the matrix W, we use *singular value decomposition* (SVD) [19]. In SVD, a matrix is factored into three matrices, i.e. $W = U \cdot \Sigma \cdot V^T$, where U and V are orthogonal matrices, and Σ is a diagonal matrix containing the singular values of W in non-increasing order. However, one issue with SVD is that there does not exist any connection between the singular values and the columns of W. To address this, we use the *permuted QR decomposition* [9, 19, 20] to map the columns of matrix W to its singular values. In case of permuted-QR decomposition, we factor the matrix W into a real orthogonal matrix Q and an upper triangular matrix R. This is an economy-size decomposition with permutation vector E, such that $W(:, E) = Q \cdot R$. The diagonal entries of R are sorted in non-increasing order. Note that this decomposition does not provide us the singular values for W directly, but via the diagonal entries of matrix R. It has been experimentally shown that the diagonal elements of R provide a close measure of the singular values of matrix W [34]. The controller parameter signal (a column in W) corresponding to the largest singular value is the most significant one (since $\|W\|_2 = \sigma_1$). Hence, we tune the controller parameter corresponding to this signal. The use of matrix decomposition in our algorithm is motivated by a similar recent application of matrix decomposition to debug Simulink models of closed-loop dynamical systems [49].

3.1.4 Complexity Analysis. The number of parameters in the controller of CCS is |P| and the length of the simulation is T (= $|\tau|$, the length of the signal ω_f). Note that we denote controller parameters by P and its corresponding signals by γ. The complexity of the function `matrix_decomposition` in line 70 is $O(T \cdot |P|^2)$. This is because the complexity of QR decomposition is $O(m \cdot n^2)$, where m and n are the number of rows and the number of columns of the matrix respectively [19]. Thus, the overall time complexity of our *select_parameter* procedure is $O(T \cdot |P|^2)$. The complexity of the *maximal_sat_spec* is $O(|\Phi|)$. The time complexity of `min_robustness` and `falsify` is linear [15] w.r.t. T, i.e., $O(T)$. The complexity of the loops in lines 24-31 and lines 34-41 is $O(T \cdot H)$.

Thus, the complexity of the *Controller Synthesis* algorithm is $O((\texttt{range}(\partial)/tol_\partial) * (T * |P|^2 + T * H + |\Phi|))$. Here $\texttt{range}(\partial)$ is the difference between the maximum and the minimum value of ∂, and H refers to our computation budget for the inner While loops in lines 24-31 and lines 34-41.

3.1.5 Soundness and Completeness. The procedure `select_parameter` depends upon the function `falsify`. The function `falsify` is *sound* because if a specification is falsified, then there indeed exists a trace that would violate the specification. However, it is not *complete* because it may not be able to find a trace that violates the specification even if there exists such a trace. Thus, even if Algorithm 1 terminates successfully, we cannot guarantee that the controller C satisfies all atomic specifications captured in Φ. However, in case there exists a falsifier that is both sound and complete (hypothetically), our controller synthesis algorithm is sound. This is because, in that case, we can remove all the root causes of falsification in the synthesized controller that we obtain from Algorithm 1. We can formalize the above discussion in the following theorem.

Theorem 3.2 (Soundness). *Suppose the falsifier employed in Algorithm 1 is both sound and complete, and Algorithm 1 terminates for specification Φ successfully. In that case, the closed-loop system with the controller returned by Algorithm 1 indeed satisfies Φ.*

$$
\begin{array}{c}\tau \quad sig\_p \quad sig\_i \quad sig\_d \\ \left(\begin{array}{cccc} \vdots & \vdots & \vdots & \vdots \\ 0.69 & 19.03 & 3.06 & -17.44 \\ 0.70 & 18.94 & 3.10 & -17.39 \\ 0.71 & 18.86 & 3.14 & -17.34 \\ 0.72 & 18.77 & 3.18 & -17.29 \\ 0.73 & 18.69 & 3.21 & -17.24 \\ 0.74 & 18.60 & 3.25 & -17.19 \\ 0.75 & 18.51 & 3.29 & -17.15 \\ \vdots & \vdots & \vdots & \vdots \end{array}\right) \end{array} . \otimes \begin{array}{c}\tau \quad \rho \\ \left(\begin{array}{cc} \vdots & \vdots \\ 0.69 & -0.0017 \\ 0.70 & -0.0013 \\ 0.71 & -0.0010 \\ 0.72 & -0.0006 \\ 0.73 & -0.0002 \\ 0.74 & 0.0002 \\ 0.75 & 0.0006 \\ \vdots & \vdots \end{array}\right) \end{array} . \implies \begin{array}{c} sig\_p \quad sig\_i \quad sig\_d \\ \left(\begin{array}{ccc} \vdots & \vdots & \vdots \\ 19.03 & 3.06 & -17.44 \\ 18.94 & 3.09 & -17.39 \\ 18.85 & 3.13 & -17.34 \\ 18.77 & 3.17 & -17.29 \\ 18.68 & 3.21 & -17.24 \\ \vdots & \vdots & \vdots \end{array}\right) \end{array} .
$$

Fig. 5. Matrices X (left), Y (middle) and W (right)

It is worth noting that Algorithm 1 is not complete. This is because there may exist controller parameters that enable the system to satisfy all the specifications, but still our algorithm may not be able to find those parameters. This is due to the incompleteness in the search procedure in lines 9-46 in the controller synthesis algorithm.

3.1.6 Example. In the example in Section 2.3, the formula $\Diamond_{[0,\tau]}\Box(|Z_t - Zref_t| < \epsilon)$ gets falsified. In function `select_parameter`, the controller parameter set γ (line 53) consists of the following set of three signals: {`sig_p`, `sig_i`, `sig_d`}, corresponding to the outputs of the `Gain` blocks containing the controller parameters.

In Figure 5, the column τ is the vector containing the time steps in the simulation and the column ρ provides the robustness values at different time steps. The matrix `W` is obtained (in line 67) by joining matrix `X` (obtained in line 58) and `Y` (obtained in line 65).

The permuted-QR decomposition of matrix `W` (line 70) provides a matrix `R` where $abs(diag(\mathtt{R}))$ is given by $[190.0303\ 43.0850\ 6.6998]^{\mathsf{T}}$. These values correspond to the singular values associated with each column of `W`, i.e., the controller parameter signals. Here we see that the singular values associated with the other controller parameters have significantly smaller values compared to that associated with the first parameter. The matrix `E` is given as `E` = [1 3 2], which indicates that column 1 in matrix `W` corresponds to the largest singular value, i.e., $\kappa = 1$. Column 1 in matrix `W` corresponds to the signal `sig_p`.

In procedure `controller_synthesis`, we get $\kappa = 1$ (line 10). The corresponding source block has parameter (Gain) K_p whose default value is 5 (robustness = -0.031). The final value of K_p that fixes the model with respect to ϕ (eq. 10) is 7.5 (robustness = +0.029).

3.2 Specification Synthesis

Generally, it may be the case that an overambitious user gives a set of atomic specifications which cannot be satisfied together. Controller synthesis w.r.t the set of specifications is not feasible in such cases, i.e., they cannot be satisfied simultaneously. In such cases, apart from giving the maximal set of satisfiable specifications and the correct *controller* C', we also provide the correct parameter values for the *unsatisfiable specification(s)* with which they become satisfiable by the closed-loop system with the controller C'.

We first convert the unsatisfiable atomic specification ϕ into a PSTL formula ϕ_p, where p refers to the parameter list $p = (p_1, \ldots, p_n)$. Then, we attempt to find the values $v(p)$ of parameters of the PSTL formula such that the STL formula $\phi_p(p = v(p))$ is satisfied w.r.t C', i.e., $\mathcal{M}_r(\mathcal{P} \circ C') \models \phi_p(p = v(p))$.

The specifications considered in our experiments are monotonic w.r.t. their parameters due to their fixed polarity (ζ), i.e., either + or −. For instance, consider the case of settling time specification

Algorithm 2: Specification Synthesis Algorithm

```
1  procedure specification_synthesis (M_r, φ_p)
2      p ← get_params(φ_p)
3      /* selecting a specification parameter with index i */
4      for i = 1 : |p| do
5          exv ← maxmin (p_i)
6          dval ← default (p_i)
7          step ← (exv − dval) / Δ
8          /* exploration starting from default val */
9          for val ← dval : step : exv do
10             ρ ← check_spec (M_r, φ_p(p_i = val))
11             if ρ > 0 then
12                 return ⟨φ_p(p_i = val)⟩
13     return UNSAT
```

ϕ_s and its parameter ϵ_1. As we relax the error band ϵ_1 (i.e. increase ϵ_1), ϕ_s becomes easier to satisfy. Thus, the polarity of ϵ_1 in ϕ_s is positive (+), i.e., $\zeta(\epsilon_1, \phi_s) = +$. For settling time, we have $\zeta(\epsilon_1, \phi_s) = +$ and $\zeta(\tau_s, \phi_s) = +$. For rise time, we have $\zeta(\tau_r, \phi_r) = +$ and $\zeta(\beta, \phi_r) = -$. For convergence, we have $\zeta(\tau_c, \phi_c) = +$ and $\zeta(\epsilon_2, \phi_c) = +$. For overshoot, we have $\zeta(\alpha, \phi_o) = +$. For smoothness, we have $\zeta(\tau_{sm}, \phi_{sp}) = -$ and $\zeta(\mu, \phi_{sp}) = +$. The monotonicity of the PSTL formulas w.r.t its parameters is important because finding δ-tight valuations is efficient for monotonic PSTL formulas, which is hard otherwise. In case the specification is not monotonic, we can use various heuristics. One such technique would be to use random restart in case our exploration gets trapped in local minima.

Algorithm 2 tunes the specification parameters such that an unsatisfiable specification for the synthesized controller becomes satisfiable. The input to this algorithm is the PSTL template of the unsat specification ϕ_p and the model $\mathcal{M}_r$, where $\mathcal{M}_r = \mathcal{P} \circ C'$. In line 2, we get the parameters p in the specification ϕ_p. In lines 4-12, we tune a parameter p_i such that the new controller C' satisfies ϕ_p for some δ-tight value $v(p_i)$, i.e., $\mathcal{M}_r \models \phi_p(p_i = v(p_i))$. In line 5, we get the extremal value (exv) for a given parameter using domain knowledge. For instance, in case of rise time, the parameter β can only be in $[0, 1]$. Similarly, the parameter τ used in different specifications cannot be more than the total simulation time. In any case, we only need one extremal value (min or max), since we start our search from the default value of the parameter and move in one direction due to the monotonicity of the specification. This is provided using `maxmin` function, which checks the robustness of specification w.r.t. the two extremal parameter values and return the one with larger robustness. In line 6, we get the default value of parameter p_i for which ϕ_p was unsatisfiable. In line 7, we get the step size by dividing the range by Δ. Here, Δ is a parameter of the algorithm that captures the maximum number of values that are examined for a parameter. It controls how large/small jump we make within this range while going from one value to the other. We have used $\Delta = 10$ for our experiments. In lines 9-12, we check different values val for specification parameter p_i systematically such that the specification $\phi_p(p_i = val)$ is satisfied for the model $\mathcal{M}_r$, i.e., $\mathcal{M}_r \models \phi_p(p_i = val)$.

3.2.1 Complexity Analysis. The complexity of Algorithm 2 is $O(|p| * \Delta)$. Here, $|p|$ is the number of parameters in the specification ϕ_p and Δ is the maximum number of values that are examined for each parameter.

Table 1. CPS model benchmarks

System	Ref.	Specification Parameters											
		ϕ_s				ϕ_r		ϕ_c		ϕ_o	ϕ_{sp}		
		δ_1	ϵ_1	τ_s	T	τ_r	β	τ_c	ϵ_2	α	τ_{sm}	δ_2	μ
QSISO	[26]	0.10	0.03	5.00	30.00	7.00	0.80	15.00	0.10	1.10	-	-	-
CC	[3]	0.10	0.01	12.00	60.00	1.00	0.90	15.00	0.10	1.05	-	-	-
AP	[10]	0.10	0.01	5.00	30.00	2.00	0.80	10.00	0.10	1.15	5.00	0.10	0.10
IP	[12]	0.10	0.05	1.00	3.00	-	-	-	-	0.50	0.10	0.10	15.00
DCM	[11]	0.10	0.04	1.00	10.00	-	-	-	-	1.15	2.00	0.10	0.50
DPC	[32]	0.10	0.001	3.00	60.00	0.90	0.85	1.00	0.15	-	-	-	-
AMC	[31]	0.10	0.05	2.00	20.00	0.40	0.95	6.00	3.50	-	8.00	0.10	0.30
Heatex	[35]	0.10	0.001	70.00	200.00	20.00	0.80	80.00	0.10	2.00	3.00	0.10	0.01
APJ	[33]	0.10	0.005	3.00	10.00	0.02	1.00	2.00	0.05	1.20	1.00	0.10	0.05
Robotarm	[42]	0.10	0.05	4.00	20.00	-	-	4.00	0.10	2.30	2.00	0.10	12.00
QMIMO	[46]	0.10	0.10	3.00	100.00	2.30	0.80	10.00	0.10	1.25	-	-	-

4 EXPERIMENTS

4.1 Implementation

Our controller synthesis algorithm relies on runtime monitoring with respect to an STL specification. We use BREACH toolbox [14] for this purpose. Alternatively, other tools such as S-Taliro [1] and AMT [40] could also be used with our algorithm. We write a wrapper Matlab script on top of the BREACH tool to implement our algorithm. In the implementation of the procedure `select_parameter`, we retrieve the time-series data of the signals using MATLAB functions. We used a machine with core *i*7 Intel processor, 8 GB RAM, Ubuntu 18.04 OS, MATLAB version *R*2020*a*, and BREACH version 1.5.2 for our experiments.

All the files related to our experiments can be found at https://github.com/iitkcpslab/SCoSyn.

4.2 Benchmarks

We carry out our experiments on eleven closed-loop control systems. The STL specifications for these systems, which are borrowed from [25], are presented in Section 2.1.4, and the default values of the parameters are presented in Table 1. Though the systems have been taken from standard sources, they turned out not to be correct with respect to several useful STL specifications. We provide a brief introduction to these systems below, which are available in the above-mentioned Github link.

4.2.1 Systems with simple Controllers.

Quadcopter-SISO (QSISO). In the QSISO [26] model, the controller generates a control command to move the quadcopter along the direction of the z-axis. The input to the controller is the desired z-coordinate. Here the specifications capture various requirements involving the current z-coordinate (Z) and the desired z-coordinate ($Zref$) of the quadcopter.

Cruise Control (CC). In the CC [3] model, we regulate the speed of an automobile. The input to the controller is the reference speed v_{ref}. For a given reference speed, the controller accelerates/decelerates the vehicle in order to maintain the desired speed.

Aircraft Pitch (AP). In the AP [10] system, an autopilot model controls the pitch of the aircraft. The basic requirement for the aircraft is to be in steady-cruise at constant altitude and velocity. The input to the controller is the reference pitch angle θ_{ref}.

Inverted Pendulum (IP). The IP [12] model consists of an Inverted Pendulum mounted to a motorized cart. The importance of this model lies in the fact that it is always unstable without control. Hence, the pendulum cannot be balanced if the cart is not moving. So, we want to balance the pendulum by applying the required force to move the cart. For this system, the force F is the

control input that moves the cart horizontally, and the angular position of the pendulum θ is the output.

DC Motor (DCM). The DCM [11] provides both rotary motion and translational motion. For translational motion, it needs to be coupled with wheels. For this system, the voltage (V) applied to the motor is the input, and the rotational speed of the shaft $\dot{\theta}$ is the output.

4.2.2 *Systems with Cascaded/Nested/Multi-Loop Controller.*

Digital pitch control of an Aircraft (DPC). The controller used in this model [32] enables us to operate an aircraft at a high angle of attack with minimal intervention of the pilot. The system has a Dryden wind gust component which is used to create perturbations in the system, making the control of this system a complex task. The controller uses the pilot's stick pitch command as the setpoint for the aircraft's pitch attitude. The controller consists of parameters K_a (alpha), K_q (pitch) and K_f (feedforward).

Airframe Model (cascaded feedback) (AMC). The autopilot of an airframe [31] controls its fin deflection. In this model, two cascaded feedback loops are used to achieve the reference acceleration. The controller's feedback loop structure uses the pitch rate (K_q) as the inner feedback loop and the vertical acceleration (K_{az}) as an outer feedback loop.

Temperature control in Heat Exchanger (Heatex). Heatex [35] uses feedback and feed-forward controllers to regulate the temperature of a chemical plant via a heat exchanger. Here, a constant temperature $setp$ (setpoint) for the tank is maintained by varying the supply of steam to the heat exchanger. The disturbance dis is caused by the variations in the temperature of the inlet flow. The temperature of the tank at a given time is denoted by $temp$.

Autopilot for Passenger Jet (APJ). The APJ [33] is a *nested* controller for a supersonic passenger jet flying at Mach 0.74 and 5000 feets. The autopilot is used to follow commands issued by the pilot. It consists of an inner loop that controls the pitch rate (Q) and an outer loop that controls the vertical acceleration (using PI). This also contains a feed-forward (F) component and a second-order roll-off filter (R) to control noise and bandwidth against unmodeled dynamics. The input to the autopilot is the vertical acceleration commands (Nzc) from the pilot, and the output is the actual acceleration (Nz).

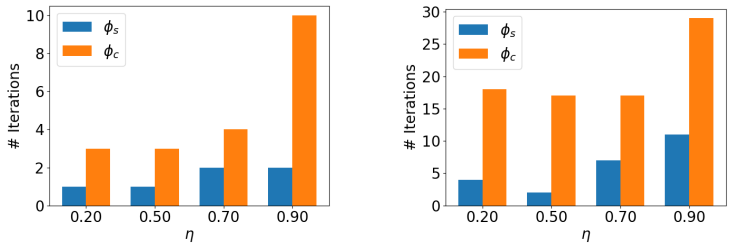
Robot-Arm. The CRS Robot Arm model [42] consists of three motors, each for waist joint, shoulder joint, and elbow joint. We have assumed that the waist joint is fixed. The motors of the shoulder and elbow are controlled using two strongly coupled PID controllers. Due to this coupling, simultaneous tuning of both these controllers is a challenging problem. Here, the input is the θ^{2md} and θ^{3md} which refers to the desired shoulder and elbow angles, respectively.

Quadcopter-MIMO (QMIMO). QMIMO [46] is one of the most complex systems for *multi-loop* feedback control. It uses six PID controllers (each for x, y, z, ϕ, θ, ψ). Hence, there are 18 controller parameters. This is an underactuated system, since we are able to move in 6-DOF with only four control inputs. The input to the system is the reference positions for x, y, z, and angle ψ. The output of the model is the translational position x, y, z and rotational position ϕ, θ, ψ. While x and y are controlled via translational controller, the variables z, ϕ, θ, ψ are controlled via attitude/altitude controller. The quadcoptor's attitude is controlled by a lower-level controller, which runs at a higher rate than the higher-level controller that controls the translational position of the quadcopter. An important point to note is that the translational movement depends upon the attitude adjustments, and hence the output of the translational controller is the input to the attitude controller.

In the examples above, some specifications were *not applicable* in certain cases (shown as "-" in Table 1). For instance, the convergence specification does not apply if no reference input signal (for

Table 2. Comparison between different falsifier algorithms in BREACH

Model	Metrics	Nelder-Mead	FMINSEARCH	CMAES	SIM-ANNEALING	FMINCON	GA
QSISO	T_{cs}(s)	238.2	261.29	141.73	244.97	385.07	331.48
	#Iterations	4	4	4	4	7	4
	Final_val (P, I, D)	37.96, 1.00, 10.00	37.96, 1.00, 10.00	37.96, 1.00, 10.00	37.96, 1.00, 10.00	56.95, 0.25, 10.00	37.96, 1.00, 10.00
CC	T_{cs}(s)	147.63	881.3	262.29	175.73	275.14	770.73
	#Iterations	5	18	6	5	4	12
	Final_val (P, I, D)	5.6953, 0.10, 1.00	0.9492, 0.0475, 0.125	0.1875, 0.10, 0.125	5.6953, 0.10, 10	3.7969, 0.10, 10	0.4219, 0.025, 0.1406
IP	T_{cs}(s)	422.12	243.16	360.27	353.34	365.90	358.38
	#Iterations	11	6	8	8	8	8
	Final_val (P, I, D)	5.22, 16.395, 66.122	5.22, 65.58, 19.5919	5.22, 16.395, 19.5919	5.22, 16.395, 19.5919	5.22, 16.395, 19.5919	5.22, 16.395, 19.5919
DCM	T_{cs}(s)	150.87	256.61	123.77	324.11	208.39	286.36
	#Iterations	7	6	4	7	5	6
	Final_val (P, I, D)	10.00, 0.7812, 1.65	7.50, 6.25, 1.65	10.00, 6.25, 1.65	10.00, 0.7812, 1.65	5.00, 3.125, 1.65	2.50, 3.125, 1.65



(a) AP (b) Q-MIMO

Fig. 6. Number of iterations for different values of η for specification ϕ_s and ϕ_c. The models used for this plot is AP (left) and QMIMO (right). The default value of η used in our experiments is $\eta = 0.5$ ($\partial_l = 0.5$ and $\partial_r = 1.5$).

the actual output) exists. Similarly, some systems, by default, have a smoother trajectory. Hence the smoothness specification is not applicable.

The default controller parameter values AP, IP, DCM and Robot Arm are set using MATLAB PID tuner by keeping a balance between performance and robustness. In case of QSISO, CC, DPC, AMC, Heatex, APJ and QMIMO, the default values are taken from their references.

4.3 Results

4.3.1 Comparison between different falsification algorithms. The Breach tool provides different optimization techniques to solve the falsification problem through finding a trace with minimal robustness. We carry out an experiment to find the efficacy of these optimization techniques in solving the controller synthesis problem we address in this paper. A comparison between different falsification algorithms is presented in Table 2. We ran the experiments on QSISO, CC, IP, and DCM to find controllers for the conjunct of all specifications. From the experimental results, we find that even though the falsifier algorithms differ significantly in some cases, our controller synthesis algorithm was robust enough to find a controller that satisfied the conjunct of specifications with each falsifier algorithm. Note that some falsifiers use randomization (like FMINCON), and hence we ran each experiment for 3 times and present the data for the worst case (in terms of the number of iterations). The number of iterations refers to the main loop of the controller synthesis algorithm, and hence it is possible that even with fewer iterations, an experiment may take more computation time due to the inefficiency of the optimization procedure. In all our experiments presented in the rest of the paper, we use `Nelder-Mead` algorithm for the falsifier as it is the default optimizer in Breach.

Table 3. Results for individual specifications. Tuned parameters are in bold font.

System	Default Value	Spec	Final Value	Iters	T_{cs}(s)
QSISO	$P = 5, I = 1, D = 10$	ϕ_s	$\mathbf{P = 7.5}$, $I = 1$, $D = 10$	1	51.85
		ϕ_r	$\mathbf{P = 25.31}$, $I = 1$, $D = 10$	2	73.65
		ϕ_c	$\mathbf{P = 7.5}$, $I = 1$, $D = 10$	1	51.85
		ϕ_o	$P = 5$, $\mathbf{I = 0.25}$, $D = 10$	1	59.90
CC	$P = 0.5, I = 0.1, D = 1$	ϕ_s	$\mathbf{P = 0.75}$, $I = 0.1$, $D = 1$	1	45.18
		ϕ_r	$\mathbf{P = 1.125}$, $I = 0.1$, $D = 1$	1	57.07
		ϕ_c	$\mathbf{P = 5.6953}$, $I = 0.1$, $D = 1$	5	107.37
		ϕ_o	$\mathbf{P = 0.75}$, $I = 0.1$, $D = 1$	1	45.80
AP	$P = 0.495, I = 0.348, D = 0.115$	ϕ_s	$P = 0.495$, $\mathbf{I = 0.174}$, $D = 0.115$	1	27.69
		ϕ_r	$\mathbf{P = 0.7425}$, $I = 0.348$, $D = 0.115$	1	38.44
		ϕ_c	$\mathbf{P = 1.67}$, $I = 0.348$, $D = 0.115$	3	59.14
		ϕ_o	$P = 0.495$, $\mathbf{I = 0.0217}$, $D = 0.115$	3	90.17
		ϕ_{sp}	$P = 0.495$, $\mathbf{I = 0.174}$, $D = 0.115$	1	29.52
IP	$P = 41.76, I = 65.58, D = 3.87$	ϕ_s	$\mathbf{P = 140.94}$, $I = 65.58$, $D = 3.87$	2	62.92
		ϕ_o	$\mathbf{P = 93.96}$, $I = 65.58$, $D = 3.87$	1	55.61
		ϕ_{sp}	$\mathbf{P = 5.22}$, $\mathbf{I = 16.395}$, $\mathbf{D = 19.591}$	8	274.66
DCM	$P = 20, I = 50, D = 1.65$	ϕ_s	$\mathbf{P = 2.5}$, $\mathbf{I = 3.125}$, $D = 1.65$	6	128.43
		ϕ_o	$P = 20$, $\mathbf{I = 25}$, $D = 1.65$	1	27.76
		ϕ_{sp}	$P = 20$, $\mathbf{I = 25}$, $D = 1.65$	1	28.65
DPC	$K_f = -3.864$, $K_\alpha = 0.677$, $K_p = 0.8156$	ϕ_s	$K_f = -3.864$, $K_\alpha = 0.677$, $\mathbf{K_p = 1.2744}$	2	62.53
		ϕ_r	$K_f = -3.864$, $K_\alpha = 0.677$, $\mathbf{K_p = 0.6117}$	1	46.71
		ϕ_c	$K_f = -3.864$, $\mathbf{K_\alpha = 1.3223}$, $K_p = 0.8156$	2	84.23
AMC	$K_{az} = 0.00027507$, $K_q = 2.7717622$	ϕ_s	$\mathbf{K_{az} = 0.00041260}$, $K_q = 2.7717622$	2	60.95
		ϕ_r	$\mathbf{K_{az} = 0.00061891}$, $K_q = 2.7717622$	3	69.96
		ϕ_c	$\mathbf{K_{az} = 0.0004126}$, $K_q = 2.7717622$	2	60.95
		ϕ_{sp}	$\mathbf{K_{az} = 0.00029374}$, $K_q = 2.7717622$	11	378.07
Heatex	$K_{ff} = 1,\ K_{fb} = 1$	ϕ_s	$K_{ff} = 1$, $\mathbf{K_{fb} = 0.5625}$	3	131.61
		ϕ_r	$K_{ff} = 1$, $\mathbf{K_{fb} = 3.375}$	2	78.94
		ϕ_c	$\mathbf{K_{ff} = 0.9492}$, $\mathbf{K_{fb} = 0.6328}$	14	374.41
		ϕ_o	$\mathbf{K_{ff} = 0.0625}$, $\mathbf{K_{fb} = 0.5}$	5	127.49
		ϕ_{sp}	$K_{ff} = 1$, $\mathbf{K_{fb} = 0.5}$	1	64.50
APJ	$P = -0.009821$, $I = -0.0297$, $F = -0.02233$, $Q = -0.2843$, $R = 4.81$	ϕ_s	$P = -0.009821$, $\mathbf{I = -0.0093973}$, $F = -0.02233$, $Q = -0.2843$, $R = 4.8100$	3	88.95
		ϕ_r	$P = -0.009821$, $I = -0.0297$, $\mathbf{F = -0.0436}$, $Q = -0.2843$, $R = 4.8100$	2	64.2
		ϕ_c	$P = -0.009821$, $I = -0.0297$, $\mathbf{F = -0.0022}$, $\mathbf{Q = -0.7295}$, $R = 4.8100$	15	498.71
		ϕ_o	$P = -0.009821$, $\mathbf{I = -0.0125}$, $\mathbf{F = -0.0171}$, $\mathbf{Q = -0.2843}$, $R = 4.8100$	5	121.4
		ϕ_{sp}	$P = -0.009821$, $I = -0.0297$, $\mathbf{F = -0.0167}$, $Q = -0.2843$, $R = 4.8100$	1	47.67
Robot Arm	$Kp_1 = 50, Ki_1 = 2, Kd_1 = 0.5,$ $Kp_2 = 50, Ki_2 = 1, Kd_2 = 2$	ϕ_s	$\mathbf{Kp_1 = 25}$, $Ki_1 = 2$, $Kd_1 = 0.5$, $Kp_2 = 50$, $Ki_2 = 1$, $Kd_2 = 2$	1	44.74
		ϕ_c	$\mathbf{Kp_1 = 17.7979}$, $Ki_1 = 2$, $Kd_1 = 0.5$, $\mathbf{Kp_2 = 37.5}$, $Ki_2 = 1$, $Kd_2 = 2$	4	109.41
		ϕ_o	$\mathbf{Kp_1 = 3.125}$, $Ki_1 = 2$, $Kd_1 = 0.5$, $\mathbf{Kp_2 = 12.5}$, $Ki_2 = 1$, $Kd_2 = 2$	6	193.06
		ϕ_{sp}	$\mathbf{Kp_1 = 3.5156}$, $Ki_1 = 2$, $Kd_1 = 0.5$, $\mathbf{Kp_2 = 3.5156}$, $Ki_2 = 1$, $Kd_2 = 2$	14	644.42
QMIMO	$X = 0.1, 0, -0.1$, $Y = 0.1, 0, -0.1$, $Z = 4, 0, -4$, $\phi = 4.5, 0, 0$, $\theta = 4.5, 0, 0$, $\psi = 10, 0, 0$	ϕ_s	$X = 0.1, 0, -0.1$, $Y = 0.1, 0, -0.1$, $\mathbf{Z = 1.6875, 0, -4}$, $\phi = 4.5, 0, 0$, $\theta = 4.5, 0, 0$, $\psi = 10, 0, 0$	2	233.38
		ϕ_r	$X = 0.1, 0, -0.1$, $Y = 0.1, 0, -0.1$, $\mathbf{Z = 3, 0, -4}$, $\phi = 4.5, 0, 0$, $\theta = 4.5, 0, 0$, $\psi = 10, 0, 0$	1	177.31
		ϕ_c	$X = 0.1, 0, -0.1$, $Y = 0.1, 0, -0.1$, $\mathbf{Z = 1.5625, 0, -4}$, $\phi = 4.5, 0, 0$, $\theta = 4.5, 0, 0$, $\psi = 10, 0, 0$	17	1005.86
		ϕ_o	$X = 0.1, 0, -0.1$, $Y = 0.1, 0, -0.1$, $\mathbf{Z = 1.6875, 0, -4}$, $\phi = 4.5, 0, 0$, $\theta = 4.5, 0, 0$, $\psi = 10, 0, 0$	2	208.26

4.3.2 Deciding the parameter values for Algorithm 1. Algorithm 1 involves parameter η (since $\partial = 1 \pm \eta$). The performance of the algorithm depends on the value used for this parameter. Here the performance is measured as the number of iterations required to tune a parameter in a falsified model. We carried out experiments on the closed-loop systems introduced in Section 4.2 and the specifications introduced in Section 2.1.4 to understand how the number of iterations to solve the parameter tuning problem varies with the value of η. Figure 6 shows the results for the systems AP and QMIMO and the specifications ϕ_s and ϕ_c. For the range of ∂, we find that the performance of the algorithm is poor for a large range of ∂. It attains the best performance for $\eta = 0.5$ and degrades as we increase η. Hence, we choose $\eta = 0.5$ for all our experiments.

4.3.3 Results for individual specifications. In Table 3, we show the results of our controller synthesis algorithm for different specifications independently for each system with simple controllers and complex controllers, respectively. The second column contains the default controller parameter values. The final value refers to the values of the parameters that fix the controller such that the specifications are satisfied. The controller parameters whose values changed in the tuning process

have been shown in bold font. In the fifth column, Iters refers to the number of iterations of the main loop in Algorithm 1. The computation times have been provided in the last column.

In what follows, we describe our observations on the execution of our algorithm while synthesizing the controllers satisfying individual specifications. Our objective is to convince the readers that our automatic controller parameter tuning procedure works in a way that can be easily explained by our understanding of how a physical process behaves under the action of its controller. This bolsters the fact that our algorithm works on the principles that govern the operation of the closed-loop control systems.

In QSISO, the controller used is a simple PID controller. The falsification of a specification, say ϕ_s, is due to the plant not being able to get into a steady state within the specified time. This is achieved by increasing the proportional gain or reduction in the derivative gain (to reduce the dampening effect). The automatic controller parameter tuner based on our algorithm also takes the same measure in finding the final value of the controller parameters that satisfy the specification. The proportional gain is increased from 5 to 7.5. Similarly, while carrying out experiments with other PID based controllers such as those in CC, IP, AP, and DCM, we observed that our automated tuner performs the tuning in the way an expert engineer would have done in the process of manual tuning.

In case of DPC, we know that increase in pitch rate K_p results in the aircraft attaining the desired angle quickly, which in turn reduces settling time. Moreover, the increase in pitch angle K_α provides more room for rotation around the pitch axis, which in turn improves convergence time. Our algorithm automatically identifies these requirements, which is evident from the fact that it increases the value of K_p to satisfy the specifications on settling time (ϕ_s), and increases the value of K_α to satisfy the specification on convergence (ϕ_c).

In case of AMC, the goal of the controller is to control the aircraft's fin deflection to help the aircraft move up/down. This can be achieved by finding correct values for the gain parameters associated with the vertical acceleration or the pitch rate. Our algorithm finds the appropriate values for the gain parameter associated with vertical acceleration K_{az} to satisfy all the specifications automatically.

In case of Heatex, the tank liquid has to be maintained at a constant temperature. The feedforward controller is responsible for the rejection of disturbances, i.e., fluctuations in the tank temperature, while the feedback controller is responsible for setpoint tracking. We observe that our algorithm is able to discover this principle automatically while finding the appropriate values for the controller parameters to satisfy various specifications. For instance, in case of ϕ_s, the violation is due to the system being much more responsive or springy. Thus, ideally, we would like to reduce the gain of the feedback controller. This is exactly what we get from our algorithm, which reduces the feedback gain K_{fb} from 1 to 0.5625.

In case of APJ, the goal is to follow the vertical acceleration issued by the pilot. Our algorithm fixes the violation of specification ϕ_c by increasing the feedforward gain and reducing the pitch rate ($F = -0.0022, Q = -0.7295$). As we know, increasing feedforward gain reduces the disturbance, and reduction in pitch rate makes the system less responsive to errors, thereby reducing oscillatory behavior. Both these factors help in achieving convergence which we also found experimentally. Similarly, in case of ϕ_s, the violation is fixed by increasing the integral gain and increasing the feedforward gain ($I = -0.0167, F = -0.0150$). We know that increasing the integral gain reduces the steady-state errors, and feedforward gain reduces the effect of disturbances. Both these factors reduce the settling time enabling satisfaction of ϕ_s.

In case of RobotArm, the specification ϕ_c is violated because both the shoulder and elbow joint could not converge with the desired configuration and keeps on oscillating around the desired configuration for a longer time. To fix this, our algorithm reduces the proportional gain of both

the controllers of shoulder and elbow joints, which is also intuitive. Our algorithm tunes the proportional gains also to satisfy the other specifications.

In the QMIMO system, we found the parameter used for repair of the specifications is K_p and K_d of the PID-controller corresponding to Z (attitude controller). The important point here is that when a specification, say ϕ_s, is falsified (Table 3), we see that the violation is along all x, y (translational) and z (attitude) directions. But, the controller synthesis involves tuning the parameters of the attitude controller only. Similar fixes we witness for the other specifications as well. It is well known that the translational movement (along x, y) of quadcopter depends upon the attitude movements (along z, ϕ, θ, ψ). Hence, tuning attitude controller is the most appropriate measure for our controller synthesis.

It may appear counter-intuitive that none of the desired specifications were satisfied by the standard controller. The reason is that the constraints involved in the specifications in all the cases were intentionally made strict about making the controller synthesis problem challenging. That is why the standard controller fails to satisfy any specification. Note that the constraints in the specifications may depend on the application and may vary from application to application. For instance, in some cases, we may want that our rise time for 90% of reference signal should be 1s. In some other case, we may want it to be 0.9s or 1.1s. A similar argument holds for other specifications.

4.3.4 Results for the set of specifications. In Table 4, we present the results of the controller synthesis algorithm for the conjunction of all specifications. Note that this is a much harder problem than the one presented in the previous subsection. Here we want to find the controller parameter values such that all the specifications are satisfied simultaneously. However, in certain cases, our algorithm is not able to find the desired controller parameter values which satisfy all the specifications. In these cases, we first fix the controller with the best parameter values found by our algorithm (denoted by C_r). Then, for the specifications which could not be satisfied along with the satisfied specifications, we synthesize the parameters of those specifications independently, so that those specifications are satisfied by the CCS with the controller C_r.

In case of QSISO, CC, IP and DCM systems, Algorithm 1 could find the suitable controller parameter values $(37.96, 1, 10)$, $(5.6953, 0.1, 1)$, $(41.76, 65.58, 66.1226)$, and $(10, 0.7812, 1.65)$, respectively, for which all the specifications were satisfied. In case of AP, DPC, AMC, Heatex, Robot Arm, and QMIMO models, the algorithm could not find the suitable controller parameter values that satisfied all the specifications. Hence, in these cases, we provide the maximal set of atomic specifications that are satisfied and also the correct parameter values for the unsatisfiable atomic specification(s) (See Table 4).

Let us explain the results in Table 4 with the example of the system DPC. The other examples can be interpreted similarly. In case of DPC model, the maximum number of specifications that could be satisfied was one (Table 4). We found two different sets of maximal satisfied specifications. In the first case, we found the best controller parameter values for which ϕ_r was satisfied, while in the second case, the best controller parameter values were the ones for which ϕ_c was satisfied. This might seem counter-intuitive that individually all specifications were satisfied for DPC (Table 3), but in case of the conjunction of these specifications, only ϕ_r and ϕ_c are satisfied. This is because when the specification $\Phi = \phi_s \wedge \phi_r \wedge \phi_c$ is falsified for the first time, the algorithm searches for parameter values that are satisfiable w.r.t Φ instead of ϕ_s, ϕ_r or ϕ_c individually. Once it moves in the direction that satisfies ϕ_r, it cannot satisfy ϕ_s. The exploration strategy of our algorithm is not static but is dynamic in the sense that it is always guided by the improvement in robustness.

In Table 4, we also present the computation time spent on the procedures `controller_synthesis` and `specification_synthesis` in Algorithm 1 and Algorithm 2 in the column T_{cs} and T_{ss}, respectively.

Table 4. Results for the set of specifications: $\Phi_{mxl} = \Phi_1$ means that the final controller satisfies all the atomic sub-specifications. For the system AP, $\Phi_{mxl} = \{\phi_s, \phi_r, \phi_c, \phi_o\}$ indicates that the final controller satisfies the atomic propositions $\phi_s, \phi_r, \phi_c, \phi_o$ directly, and ϕ_{sp} SAT @ $\mu = 0.59$ indicates that the atomic specification ϕ_{sp} gets satisfied by the final controller once the value of its parameter μ is updated to 0.59. The tuned parameters have been shown in bold font.

System	Specification	Default Value	Final Value	Iters	Result	Computation Time (s)	
						T_{cs}	T_{ss}
QSISO	$\Phi_1 = \phi_s \wedge \phi_r \wedge \phi_c \wedge \phi_o$	$P = 5, I = 1, D = 10$	$\mathbf{P = 37.96}, I = 1, D = 10$	4	$\Phi_{mxl} = \Phi_1$	238.20	-
CC	$\Phi_2 = \phi_s \wedge \phi_r \wedge \phi_c \wedge \phi_o$	$P = 0.5, I = 0.1, D = 1$	$\mathbf{P = 5.6953}, I = 0.1, D = 1$	5	$\Phi_{mxl} = \Phi_2$	147.63	-
AP	$\Phi_3 = \phi_s \wedge \phi_r \wedge \phi_c \wedge \phi_o \wedge \phi_{sp}$	$P = 0.49, I = 0.348, D = 0.115$	$\mathbf{P = 1.0622}, \mathbf{I = 0.23627}, \mathbf{D = 0.35095}$	54	$\Phi_{mxl} = \{\phi_s, \phi_r, \phi_c, \phi_o\}$ ϕ_{sp} SAT @ $\mu = 0.59$	102.32	29.18
			$\mathbf{P = 0.93358}, \mathbf{I = 0.2367}, \mathbf{D = 0.35095}$	58	SAT= $\{\phi_s, \phi_r, \phi_o, \phi_{sp}\}$ ϕ_c SAT @ $\tau_2 = 16$		55.06
IP	$\Phi_4 = \phi_s \wedge \phi_o \wedge \phi_{sp}$	$P = 41.76, I = 65.58, D = 3.87$	$\mathbf{P = 5.22}, \mathbf{I = 16.395}, \mathbf{D = 19.5919}$	8	$\Phi_{mxl} = \Phi_4$	436.83	-
DCM	$\Phi_5 = \phi_s \wedge \phi_o \wedge \phi_{sp}$	$P = 20, I = 50, D = 1.65$	$\mathbf{P = 10}, \mathbf{I = 0.7812}, D = 1.65$	7	$\Phi_{mxl} = \Phi_5$	150.87	-
DPC	$\Phi_6 = \phi_s \wedge \phi_r \wedge \phi_c$	$K_f = -3.864, K_\alpha = 0.677,$ $K_p = 0.8156$	$K_f = -3.864, K_\alpha = 0.677,$ $\mathbf{K_p = 0.4588}$	3	$\Phi_{mxl} = \{\phi_r\}$ ϕ_s SAT @ $\epsilon_1 = 0.0082$, ϕ_c SAT @ $\epsilon_2 = 0.42$	345.39	88.39
			$\mathbf{K_f = -5.5017}, K_\alpha = 0.677,$ $K_p = 0.8156$	8	$\Phi_{mxl} = \{\phi_c\}$ ϕ_s SAT @ $\epsilon_1 = 0.0028$, ϕ_r SAT @ $\beta = 0.79$		40.19
AMC	$\Phi_7 = \phi_s \wedge \phi_r \wedge \phi_c \wedge \phi_{sp}$	$K_{az} = 0.00027507,$ $K_q = 2.7717622$	$\mathbf{K_{az} = 0.00052221},$ $K_q = 2.7717622$	6	$\Phi_{mxl} = \{\phi_s, \phi_r, \phi_c\}$ ϕ_{sp} SAT @ $\mu = 100$	697.28	55.82
Heatex	$\Phi_8 = \phi_s \wedge \phi_r \wedge \phi_c \wedge \phi_o \wedge \phi_{sp}$	$K_{ff} = 1, K_{fb} = 1$	$\mathbf{K_{ff} = 0.4746}, \mathbf{K_{fb} = 0.5625}$	12	$\Phi_{mxl} = \{\phi_s, \phi_o\}$ ϕ_r SAT @ $bt = 0.16$ ϕ_c SAT @$\epsilon_2 = 0.37$ ϕ_{sp} SAT @ $\mu = 1$	1802.33	275.75
			$\mathbf{K_{ff} = 0.9010}, \mathbf{K_{fb} = 0.5625}$	17	$\Phi_{mxl} = \{\phi_s, \phi_c\}$ ϕ_r SAT @ $bt = 0.16$ ϕ_o UNSAT $\forall \alpha$ ϕ_{sp} SAT @ $\mu = 1$		60.00
			$\mathbf{K_{ff} = 0.2253}, \mathbf{K_{fb} = 0.4746}$	29	$\Phi_{mxl} = \{\phi_o, \phi_{sp}\}$ ϕ_s SAT @ $\epsilon = 0.0019$ ϕ_r SAT @ $\beta = 0.14$ ϕ_c SAT @ $\epsilon = 0.64$		89.42
APJ	$\Phi_9 = \phi_s \wedge \phi_r \wedge \phi_c \wedge \phi_o \wedge \phi_{sp}$	$P = -0.009821, I = -0.0297,$ $F = -0.02233, Q = -0.2843,$ $R = 4.81$	$P = -0.009821, \mathbf{I = -0.0167},$ $\mathbf{F = -0.0051}, Q = -0.2843,$ $R = 4.81$	3	$\Phi_{mxl} = \{\phi_s, \phi_o, \phi_{sp}\}$ ϕ_r SAT @ $\beta = 0.5$ ϕ_c SAT @ $\tau_2 = 8, \epsilon_2 = 0.1$	278.22	83.64
Robot Arm	$\Phi_{10} = \phi_s \wedge \phi_c \wedge \phi_o \wedge \phi_{sp}$	$Kp_1 = 50, Ki_1 = 2,$ $Kd_1 = 0.5, Kp_2 = 50,$ $Ki_2 = 1, Kd_2 = 2$	$\mathbf{Kp_1 = 75}, Ki_1 = 2,$ $Kd_1 = 0.5, Kp_2 = 50,$ $Ki_2 = 1, Kd_2 = 2$	2	$\Phi_{mxl} = \phi_s, \phi_c$ ϕ_o SAT @ $\alpha = 2.53$ ϕ_{sp} SAT @ $\mu = 12.88$	698.02	51.45
QMIMO	$\Phi_{11} = \phi_s \wedge \phi_r \wedge \phi_c \wedge \phi_o$	$X = (0.1, 0, -0.1), Y = (0.1, 0, -0.1),$ $Z = (4, 0, -4),$ $\phi = (4.5, 0, 0), \theta = (4.5, 0, 0),$ $\psi = (10, 0, 0)$	$X = (0.1, 0, -0.1), Y = (0.1, 0, -0.1),$ $\mathbf{Z = (1.5625, 0, -6.4072)},$ $\phi = (4.5, 0, 0), \theta = (4.5, 0, 0),$ $\psi = (10, 0, 0)$	30	$\Phi_{mxl} = \{\phi_s, \phi_c, \phi_o\}$ ϕ_r SAT @ $\tau_1 = 6.44$	1018.33	148.31

4.3.5 Other Specifications. The specifications considered in the paper are standard specifications for capturing the correctness and performance of feedback controllers. We carried out exhaustive experiments with these specifications as described before. However, our technique can handle any monotonic STL specifications. Below we present results with a few such specifications.

- `Nested-Bounded Operators:`
  - `Eventually-Always:` For the AP model, we modify the convergence property to make always operator as bounded, i.e., $\phi_c = \Diamond_{[0,3]} \Box_{[0,10]} (|\theta_t - \bar{\theta}_t| < \epsilon)$. This means that within 3s, θ should reach and remain converged to $\bar{\theta}$ for 10s. Our tool synthesizes the new controller $(1.67, 0.348, 0.115)$ within 3 iterations. This is expected since $\Box$ without bounded interval is a stricter operator than $\Box$ with bounded interval.
  - `Infinitely-Often:` Consider the AP model with the following specification: $\Box_{[0,10]} \Diamond_{[0,3]} (|\theta_t - \bar{\theta}_t| < \epsilon)$, where $\epsilon = 0.1$. This specification means that θ should converge to $\bar{\theta}$ within every 3s and this behavior should continue for 10s. In this case , we get a falsification for the default controller. Using our tool we are able to synthesize the new controller $(0.495, 0.522, 0.115)$ within 2 iterations so that it satisfies the specification.

Table 5. Experimental Results with Random Search and Simulated Annealing for the set of specifications. Timeout = 3600s

System	Spec	Random Search (RS)				Simulated Annealing (SA)			
		Iters	T(s)	Final Value	Φ_{mxl}	Iters	T(s)	Final Value	Φ_{mxl}
QSISO	Φ_1	36	558.01	$P = 7.81, I = 1.56, D = 4.21$	Φ_1	42	805.91	$P = 213.8, I = 4.187, D = 130.79$	Φ_1
CC	Φ_2	26	947.183	$P = 0.75, I = 0.24, D = 0.17$	Φ_2	94	Timeout	$P = 46.49, I = 3.59, D = 35.15$	$\{\phi_s,\phi_r,\phi_o\}$
AP	Φ_3	99	Timeout	$P = 0.495, I = 0.5022, D = 0.8733$	$\{\phi_s,\phi_c,\phi_{sp}\}$	11	301.13	$P = 31.669, I = 0.025, D = 10.749$	Φ_3
IP	Φ_4	91	Timeout	$P = 41.76, I = 17.50, D = 29.38$	$\{\phi_s, \phi_o\}$	18	457.43	$P = 1.77, I = 39.18, D = 24.72$	Φ_4
DCM	Φ_5	84	Timeout	$P = 20, I = 0.78, D = 1.65$	$\{\phi_o, \phi_{sp}\}$	96	Timeout	$P = 10.72, I = 17.36, D = 6.31$	$\{\phi_o,\phi_{sp}\}$
DPC	Φ_6	71	Timeout	-	$\{\}$	82	Timeout	$K_f = -0.37, K_\alpha = 57.28, K_p = 0.078$	$\{\phi_r\}$
AMC	Φ_7	80	Timeout	$K_{az} = 0.0005, K_q = 2.7718$	$\{\phi_s, \phi_r, \phi_c\}$	94	Timeout	$K_{az} = 0.0037, K_q = 5.93$	$\{\phi_r\}$
Heatex	Φ_8	80	Timeout	$K_{ff} = 0.3379, K_{fb} = 0.4746$	$\{\phi_o, \phi_{sp}\}$	97	Timeout	$K_{ff} = 62.19, K_{fb} = 0.01$	$\{\phi_{sp}\}$
APJ	Φ_9	78	Timeout	-	$\{\}$	95	Timeout	$P = -1.77, I = -1.02, F = -0.58,$ $Q = -0.38, R = 22.32$	$\{\phi_r,\phi_c,\phi_{sp}\}$
RobotArm	Φ_{10}	77	Timeout	$Kp_1 = 144.5, Ki_1 = 66.35, Kd_1 = 0.39,$ $Kp_2 = 50, Ki_2 = 1, Kd_2 = 2$	$\{\phi_s\}$ $\{\phi_s\}$	89	Timeout	$Kp_1 = 95.73, Ki_1 = 6.64, Kd_1 = 3.02,$ $Kp_2 = 102.49, Ki_2 = 7.994, Kd_2 = 3.88$	$\{\phi_s\}$ $\{\phi_s\}$
QMIMO	Φ_{11}	67	Timeout	-	$\{\}$	75	Timeout	-	$\{\}$

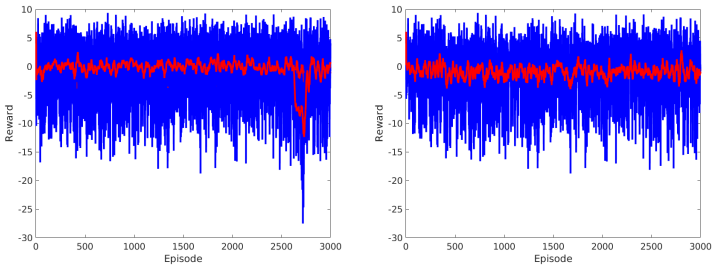
- `Reach-Avoid`: Consider the RobotArm model with the following specification: $\phi_1 U_{[0,\tau]} \phi_2$, where $\phi_1 = ((\theta_t^{2m} < \alpha \cdot \theta_t^{2md}) \wedge (\theta_t^{3m} < \alpha \cdot \theta_t^{3md}))$, $\phi_2 = \Box((|\theta_t^{2m} - \theta_t^{2md}| < \epsilon) \wedge (|\theta_t^{3m} - \theta_t^{3md}| < \epsilon))$, where $\tau = 4$, $\alpha = 2.4$ and $\epsilon = 0.1$. Here, θ^{2m} and θ^{2md} refers to the actual and the desired shoulder angles respectively. Similarly, θ^{3m} and θ^{3md} refer to the actual and the desired elbow angles respectively. The specification $\phi_1 U_{[0,\tau]} \phi_2$ means that the arm should not exceed a given bound until, within τ second, it converges with the desired value. For this specification, we get a falsification for the default controller. Using our tool we are able to synthesize the new controller $(75, 2, 0.5, 50, 1, 2)$ within 3 iterations such that it satisfies the specification.

4.4 Numerical Comparison with Alternative Techniques

4.4.1 Comparison w.r.t Random Search and Simulated annealing. In Table 5, we provide detailed results for Random Search (RS) and Simulated Annealing (SA) for all the benchmarks on the combined specifications for the sake of comparison with the results of our algorithm presented in Table 4. The comparison with RS is to show the significance of using the matrix decomposition technique in implementing the `select_parameter` function at line 10 in the controller synthesis algorithm (Algo 1). To implement RS, we implement the `select_parameter` function to generate a random sequence of parameter indexes without changing any other part of Algorithm 1. Thus, the superior performance of our algorithm to that of RS establishes the efficacy of the matrix decomposition based parameter tuning. For SA, we use the MATLAB function `simulannealbnd` and pass the default value and bounds as used in our controller synthesis algorithm as the initial values and the lower and upper bounds of the parameters, respectively. For this comparison, we consider only Algorithm 1, and implement a wrapper function to generate a controller (provided in the Final Value columns) with the maximal satisfiable subset (provided in columns Φ_{mxl}) for both RS and SA algorithms. As the maximum time taken by our algorithm is 1802s (for Heatex, see Table 4), we set a timeout of 3600s for these experiments.

As we see from Table 5, RS can find controllers satisfying all the atomic sub-specifications only for QSISO and CC. However, the computation time is significantly more than our algorithm. For DPC, APJ, and QMIMO, RS could not find a controller satisfying any of the atomic sub-specifications. For AP, IP, DCM, and RobotArm, the maximal set of sub-specifications Φ_{mxl} obtained within timeout is smaller than the one found by our algorithm, and for AMC and Heatex, RS finds the same Φ_{mxl} as our algorithm. In summary, our algorithm outperforms RS conclusively on all the benchmarks.

Coming to the comparison with Simulated Annealing (SA), it can find controllers satisfying all the atomic sub-specifications for QSISO and AP and IP. Note that our method did not succeed in finding a controller satisfying all the sub-specifications for AP. On the other hand, for QMIMO, SA



(a) 2-layer actor and critic (b) 3-layer actor and critic

Fig. 7. Training an RL-based controller for QSISO model

could not find a controller satisfying any of the atomic sub-specifications. For CC, DCM, AMC, Heatex, and RobotArm, the maximal set of sub-specifications Φ_{mxl} obtained within timeout is smaller than the ones found by our algorithm, and for DPC and APJ, SA finds the same Φ_{mxl} as our algorithm.

To understand the search strategy and computation effort required by the three methods under consideration, consider the results for QSISO in Table 4 and Table 5. For QSISO, we got a solution with all three algorithms - CS, RS, and SA. However, note that our algorithm needed to tune only a single parameter as compared to the tuning of multiple parameters in RS and SA. In every iteration of our algorithm, we tune that parameter which leads to the maximum improvement in robustness. This enables us to reach the solution faster in fewer iterations. However, as our algorithm is based on local search, it may not find a controller satisfying all the atomic sub-specifications even though there exists one. This fact is observed in case of AP, where SA, due to the global search strategy, could find a controller satisfying all the atomic sub-specifications, but our algorithm could not.

4.4.2 Comparison with ML/RL based controllers. Motivated by the work by Lillicrap et al. on synthesizing continuous controllers using Deep Reinforcement Learning (DRL) [28], we attempt to synthesize a DRL based controller to satisfy a set of STL specifications. For our experiments, we train a DDPG (Deep Deterministic Policy Gradient) agent [28] to generate the desired control for the QSISO model. In this context, the DDPG agent is the RL-based controller. In this approach, we need to generate the rewards online (at each time-step) and hence we compute the robustness of satisfaction of the conjunct of STL specifications online using BREACH. We use the specification $\Phi_1' = \phi_r \wedge \phi_c \wedge \phi_o$ (specifications like ϕ_s are not supported by BREACH (Online)).

We run the experiment for 3000 episodes. Each episode consists of 30 time-steps. For our experiments, we use the following parameter values: learning rate (actor) = 0.01, learning rate (critic) = 0.1, discount factor = 1, and minibatch size = 64. We use two different DDPG agent configurations - in the first one, both actor and critic DNNs have two hidden layers, and in the second one, both actor and critic have three layers. The training finished for the two scenarios in 4391.17s and 5541.40s respectively, without getting a convergence (refer Figure 7). We represent the Episode reward with blue points and the average reward (of the last 20 episodes) with red points. A desirable RL-based controller would have generated controls which would have led to the satisfaction of the specification ϕ_1 (and hence positive reward) at convergence. This experimental result demonstrates that Deep RL is not an alternative to our controller synthesis technique.

5 RELATED WORK

Controller synthesis for complex nonlinear systems has been one of the key challenges faced by the CPS community. The goal of controller synthesis is to design a mechanism for generating the desired inputs for the plant such that it satisfies a given set of specifications. Of late, this area has

received a lot of interest among the CPS research community. Many powerful tools have been developed for the same recently, such as Pessoa [36, 45], CoSyMA [39], LTLMop [17], Tulip [51], etc.

Recently, various research work have been carried out to synthesize controllers from specifications. In [29], the authors have addressed the problem of synthesizing controllers that satisfy multiple specifications. In this work, the choice of controller depends upon the type of plant (discrete or continuous). The work in [47] is on SMT-based PID controller synthesis technique for stochastic hybrid systems providing safety guarantees. In [13], the goal is to design a controller that satisfies a set of hard LTL formulas and minimally violates a set of soft LTL formulas. An automatic correct-by-construction synthesis technique for stochastic linear dynamical systems satisfying STL specifications is presented in [22]. As evident from the above discussion, the previous work deal with either the stability or the safety properties for controller synthesis or address the problem of controller synthesis for a special class of systems. On the other hand, our technique can deal with general nonlinear systems with arbitrary STL specifications.

Many techniques involving Machine Learning (ML)/ Reinforcement Learning (RL) have recently been applied to controller synthesis. In [50], a DNN controller is generated for a quadrotor model through supervised learning. Supervised learning is not applicable to our problem as we attempt to synthesize a superior controller to the default controller. Thus, training data generated from a default controller would not lead to a controller that could satisfy a quantitative STL specification, as the default controller itself does not satisfy the specification. Reinforcement learning is an attractive technique for synthesizing continuous control for dynamical systems [28]. In [21], RL is used to synthesize controllers for nonlinear systems like quadrotors. Another RL-based approach is presented in [8] for MDPs that satisfy an LTL specification. These techniques are similar to ours in the sense that they need to tune parameters involved in a deep neural network (weights and biases). However, as our experimental results in Section 4.4.2 demonstrates, synthesizing RL-based controllers is computationally too expensive.

In [18], the authors demonstrate how to synthesize model-predictive controllers to satisfy STL specifications. On the other hand, our focus is on synthesizing parameterized static controllers for a group of STL specifications, which do not need to solve complex optimization problems online. The authors in [18] also address the specification repair problem. For a given STL specification, the authors first check if the constraints in the specification are satisfiable by converting them into MILP constraints and solving them. In case the constraints are not satisfiable, a set of atomic predicates in the specification are identified and then repaired using slack variables. Synthesizing the parameters involved in the template STL specifications for closed-loop control systems has also been addressed in [23] and [24].

6 CONCLUSION

In this paper, we have presented a novel framework for controller synthesis for closed-loop control systems. The proposed controller synthesis framework is implemented as fully automated software. Our synthesis approach is guided by a set of complex STL specifications, which has not been considered in contemporary work. For such specifications, our tool is capable of dealing with a rich set of controllers having multiple independent and nested control loops. However, a significant limitation of our techniques is that it is applicable only if the controller contains tunable parameters (e.g., Gains, Transfer Function, etc.). Unlike the work by Raman et al. [43, 44], our tool is not readily applicable to synthesizing model predictive controllers for STL specifications.

In the future, we plan to explore the applicability of our technique to other types of controllers, such as Neural-Network based controller, which contains a large number of parameters. We will attempt to keep the size of the neural network smaller without compromising its correctness

with respect to the formal specifications. Reducing the size of neural networks will be useful in implementing them as embedded software.

ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for their detailed review and insightful comments on the originally submitted manuscript. The authors also thank Alexandre Donzé for his support in working with the BREACH Toolbox. The first author acknowledges the Prime Minister's Research Fellowship from the Government of India for supporting this research. This research work was carried out as part of the FMSAFE project supported by MHRD IMPRINT.

REFERENCES

[1] Y. Annpureddy, C. Liu, G. E. Fainekos, and S. Sankaranarayanan. 2011. S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems. In *TACAS*. 254–257.
[2] E. Asarin, A. Donzé, O. Maler, and D. Nickovic. 2011. Parametric Identification of Temporal Properties. In *RV*. 147–160.
[3] K. J. Astrom and R. M. Murray. 2008. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, USA.
[4] D. P. Atherton. 2000. Autotuning of PID Controllers: Relay Feedback Approach. *Int. J. Adapt. Control Signal Process.* 14, 5 (2000), 559–562.
[5] E. Bartocci, T. Ferrère, N. Manjunath, and D. Ničković. 2018. Localizing Faults in Simulink/Stateflow Models with STL. In *HSCC*. ACM, 197–206.
[6] V. Blondel and J. N. Tsitsiklis. 1997. NP-Hardness of Some Linear Control Design Problems. *SIAM J. Control Optim.* 35, 6 (1997), 2118–2127.
[7] S. Bogomolov, C. Mitrohin, and A. Podelski. 2010. Composing Reachability Analyses of Hybrid Systems for Safety and Stability. In *ATVA*. 67–81.
[8] A. K. Bozkurt, Y. Wang, M. M. Zavlanos, and M. Pajic. 2020. Control Synthesis from Linear Temporal Logic Specifications using Model-Free Reinforcement Learning. In *ICRA*. IEEE, 10349–10355.
[9] T. F. Chan. 1987. Rank revealing QR factorizations. *Linear Algebra Appl.* 88-89 (1987), 67 – 82.
[10] CTMS. [n.d.]. *Aircraft Pitch*. http://ctms.engin.umich.edu/CTMS/index.php?example=AircraftPitch&section=SimulinkModeling
[11] CTMS. [n.d.]. *DC Motor*. http://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SimulinkModeling
[12] CTMS. [n.d.]. *Inverted Pendulum*. http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=SimulinkModeling
[13] R. Dimitrova, M. Ghasemi, and U. Topcu. 2020. Reactive synthesis with maximum realizability of linear temporal logic specifications. *Acta Informatica* 57, 1-2 (2020), 107–135.
[14] A. Donzé. 2010. Breach: A Toolbox for Verification and Parameter Synthesis of Hybrid Systems. In *CAV*. 167–170.
[15] A. Donzé, T. Ferrère, and O. Maler. 2013. Efficient Robust Monitoring for STL. In *CAV*. 264–279.
[16] E.A.Lee and S.A.Seshia. 2017. *Introduction to Embedded Systems - A Cyber-Physical Systems Approach, Second Edition*. MIT Press.
[17] C. Finucane, G. Jing, and H. Kress-Gazit. 2010. LTLMoP: Experimenting with language, Temporal Logic and robot control. In *IROS*. 1988–1993.
[18] S. Ghosh, D. Sadigh, P. Nuzzo, V. Raman, A. Donzé, A. L. Sangiovanni-Vincentelli, S. S. Sastry, and S. A. Seshia. 2016. Diagnosis and Repair for Synthesis from Signal Temporal Logic Specifications. In *HSCC*. 31–40.
[19] G. H. Golub and Charles F. V. Loan. 1996. *Matrix Computations* (3 ed.). Johns Hopkins.
[20] Y. P. Hong and C. T. Pan. 1992. Rank-revealing QR factorizations and the singular value decomposition. *Math. Comp.* 58, 197 (1992), 213–232.
[21] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter. 2017. Control of a Quadrotor With Reinforcement Learning. *IEEE Robotics and Automation Letters* 2, 4 (2017), 2096–2103.
[22] S. Jha, S. Raj, S. K. Jha, and N. Shankar. 2018. Duality-Based Nested Controller Synthesis from STL Specifications for Stochastic Linear Systems. In *FORMATS*. 235–251.
[23] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia. 2015. Mining Requirements From Closed-Loop Control Models. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 34, 11 (2015), 1704–1717.
[24] G. Juniwal, A. Donzé, J. C. Jensen, and S. A. Seshia. 2014. CPSGrader: Synthesizing temporal logic testers for auto-grading an embedded systems laboratory. In *EMSOFT*. 24:1–24:10.

[25] J. Kapinski, X. Jin, J. Deshmukh, A. Donze, T. Yamaguchi, H. Ito, T. Kaga, S. Kobuna, and S. Seshia. 2016. ST-Lib: A Library for Specifying and Classifying Model Behaviors. In *SAE Technical Paper*. SAE International.

[26] I. M. Khairuddin, A. Majeed, A. Lim, J. A. Jizat, and A. A. Jaafar. 2014. Modelling and PID Control of a Quadrotor Aerial Robot. In *Manufacturing Engineering (Advanced Materials Research, Vol. 903)*. 327–331.

[27] R. Koymans. 1990. Specifying Real-time Properties with Metric Temporal Logic. *Real-Time Systems* 2, 4 (1990), 255–299.

[28] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *ICLR*.

[29] J. Lygeros, C. Tomlin, and S. Sastry. 1997. Multiobjective hybrid controller synthesis. In *Hybrid and Real-Time Systems*, O. Maler (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 109–123.

[30] O. Maler and D. Nickovic. 2013. Monitoring properties of analog and mixed-signal circuits. *STTT* 15, 3 (2013), 247–268.

[31] MathWorks. [n.d.]. *Cascaded Multiloop Feedback Design.* https://www.mathworks.com/help/control/ug/cascaded-multi-loopmulti-compensator-feedback-design.html.

[32] MathWorks. [n.d.]. *Designing a High Angle of Attack Pitch Mode Control.* https://www.mathworks.com/help/simulink/slref/designing-a-high-angle-of-attack-pitch-mode-control.html

[33] MathWorks. [n.d.]. *Fixed-Structure Autopilot for a Passenger Jet.* https://www.mathworks.com/help/control/ug/fixed-structure-autopilot-for-a-passenger-jet.html

[34] MathWorks. [n.d.]. *QR decomposition.* https://in.mathworks.com/help/matlab/ref/qr.html.

[35] MathWorks. [n.d.]. *Temperature Control in a Heat Exchanger.* https://www.mathworks.com/help/control/ug/temperature-control-in-a-heat-exchanger.html

[36] M. Mazo, A. Davitian, and P. Tabuada. 2010. PESSOA: A Tool for Embedded Controller Synthesis. In *CAV*. 566–569.

[37] L. Meier, D. Honegger, and M. Pollefeys. 2015. PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. *ICRA*, 6235–6240.

[38] P. M. Meshram and R. G. Kanojiya. 2012. Tuning of PID controller using Ziegler-Nichols method for speed control of DC motor. In *ICAESM*. 117–122.

[39] S. Mouelhi, A. Girard, and G. Gössler. 2013. CoSyMA: A Tool for Controller Synthesis Using Multi-Scale Abstractions. In *HSCC*. 83–88.

[40] D. Nickovic and O. Maler. 2007. AMT: A Property-based Monitoring Tool for Analog Systems. In *FORMATS*. 304–319.

[41] A. Pnueli. 1977. The Temporal Logic of Programs. In *SFCS*. 46–57.

[42] Quanser Real-Time Control (QUARC. [n.d.]. *CRS Robot Arm.* http://quanser-update.azurewebsites.net/quarc/documentation/quarc_using_devices_robots.html#catalyst

[43] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia. 2015. Reactive Synthesis from Signal Temporal Logic Specifications. In *HSCC*. 239–248.

[44] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia. 2014. Model predictive control with signal temporal logic specifications. In *CDC*. 81–87.

[45] P. Roy, P. Tabuada, and R. Majumdar. 2011. Pessoa 2.0: A Controller Synthesis Tool for Cyber-Physical Systems. In *HSCC*. 315–316.

[46] W. Selby. [n.d.]. *Simulating a 3DRobotics ArduPilot based quadrotor.* https://www.wilselby.com/research/arducopter/controller-design/

[47] F. Shmarov, N. Paoletti, E. Bartocci, S. Lin, S. A. Smolka, and P. Zuliani. 2017. SMT-based Synthesis of Safe and Robust PID Controllers for Stochastic Hybrid Systems. In *Hardware and Software: Verification and Testing*. 131–146.

[48] A. Silberschatz, H. Korth, and S. Sudarshan. 2005. *Database Systems Concepts* (5 ed.). McGraw-Hill, Inc., USA.

[49] N. K. Singh and I. Saha. 2020. Specification-Guided Automated Debugging of CPS Models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 11 (2020), 4142–4153.

[50] P. Varshney, G. Nagar, and I. Saha. 2019. DeepControl: Energy-Efficient Control of a Quadrotor using a Deep Neural Network. In *IROS*. 43–50.

[51] T. Wongpiromsarn, U. Topcu, N. Ozay, Huan Xu, and R. M. Murray. 2011. TuLiP: A Software Toolbox for Receding Horizon Temporal Logic Planning. In *HSCC*. 313–314.

[52] M. Zhuang and D. P. Atherton. 1993. Automatic tuning of optimum PID controllers. *IEE Proceedings D - Control Theory and Applications* 140, 3 (1993), 216–224.