A Static Checker for Reference Tracking Systems via Laplace Transform and Transfer Functions

Zheng Cheng, Dominique Méry

A Static Checker for Reference Tracking Systems via Laplace Transform and Transfer Functions *

Zheng Cheng Dominique Méry

LORIA, Université de Lorraine, France.

July 5, 2023

Abstract

We propose a static checker, based on the Laplace transform, for checking reference tracking system designs against their performance and safety requirements. It aims to filter out designs that have obvious requirement violations. This is to prevent users from performing more expensive evaluation tasks (e.g., simulation, model checking, or theorem proving) until those violations are reviewed or fixed. In the process, our checker depends on domain knowledge of the Laplace transform to represent each design into a mathematical model, namely the transfer function. Then, it derives time-domain metrics and interprets them as first-order formulas over real numbers. By doing so, we can formulate a proof obligation for checking requirement violations of a design in the Z3 SMT solver. We evaluated our approach on 10 designs from the literature and textbooks to demonstrate its practical usage and identify its limitations.

1 Introduction

Hybrid systems become prevalent in the next generation of safety-critical applications such as adaptive cruise control and medical devices embedded into human bodies. Among them, reference tracking systems are hybrid systems that regulate the system state around a given reference, which are our focus in

*This work is supported by the grant ANR-17-CE25-0005 (The DISCONT Project `http://discont.loria.fr`) from the Agence Nationale de la Recherche (ANR). We thank reviewers for their comments and their ideas for improving the document. We have made the paper publicly available on July 5, 2023and are happy with any feedback or question or comment.

this work. They generally do not contain discrete jumps and usually are parts of a complex hybrid system design.

It is important to ensure reference tracking systems behave as expected before in production or design complex hybrid systems that depend on them. This is usually done by: 1) specifying expected system behaviors via system requirements such as performance or safety requirements. 2) evaluating the designs to ensure they meet the specified requirements via simulation [2, 8, 11], model checking [14], or theorem proving approaches [21].

While existing evaluation approaches have shown to be helpful, we argue that it is beneficial to have a static checker, which quickly and statically analyzes for obvious requirement violations without much effort. Intuitively, this would prevent users from performing more expensive evaluation tasks until reported violations are reviewed or fixed. Moreover, users would gain more confidence to develop or evaluate hybrid systems based on the checked reference tracking systems.

Generally, control engineers apply the Laplace transform to design robust systems or tune their performance. In this work, we exploit the Laplace transform to develop a static checker for reference tracking systems. It aims to check violations of: 1) performance requirements (specifying the ability to follow reference signals in terms of time-domain metrics such as overshoot, peak time), and 2) safety requirements (specifying whether initial safe states would reach unsafe states). To do so, it:

- depends on domain knowledge of the Laplace transform to represent each design into a mathematical model, namely the transfer function.
- estimates time-domain metrics (e.g., overshoot, peak time) and interprets the definitions and properties of time-domain metrics using first-order formulas over real numbers.
- formulates proof obligations (POs) and sends them to an SMT solver to check performance or safety requirement violations.

We have implemented our static checker as a `Python` program, which discharges POs using the `Z3` SMT solver [7]. We evaluate it against 10 examples from the literature and textbooks. The results show that our checker can: 1) be applied to designs that are parameterized by either numerical or symbolic values. 2) identify the requirement violations in a given design. 3) automatically provide static checking results and counterexamples for the user to reproduce requirement violations in the design. Due to non-linear non-polynomial real arithmetic involved in the time-domain metrics estimation, we abstract the estimation formulas with their boundary conditions for conclusive checking results. Then, we implement an automated calibration of counterexamples to compensate for the loss of precision induced by this abstraction. The artifacts in this work can be found at our online repository: `http://github.com/veriatl/checkhybrid/`.

2 Motivation

To motivate our approach, let us study a car position tracking system[1]. The goal is to design a controller that adjusts the acceleration to drive the car to the desired reference position.

In [13], a typical workflow for such design starts by deriving its requirements (e.g., stability, performance, or safety requirements). For example, assuming p represents the time-series of the car position, we might want our design to satisfy the following safety requirement: $\forall t \cdot p(t) < r$ (i.e., the car position p never exceeds the reference position r).

Then, a design is built to satisfy the requirements. For example, a design for the car position tracking system is shown by Fig. 1 as a block diagram. Each block in the diagram represents a component, which could contain dynamics to give its input/output relationship. Then, we can show how signals flow between components by connecting these blocks using arrows.

As we can see from Fig. 1, the designed system is modeled as a closed-loop system. At each cycle, the component C produces an acceleration a to the component V. Then, component V produces velocity v under the differential equation $\dot{v} = a$. Next, the produced velocity goes to the component P to produce current car position p as the system output, under the differential equation $\dot{p} = v$. The system output is measured, and then compared with the system input, i.e., the desired position reference r, to produce an error term e. The error is then fed back to the controller C to compute the acceleration to drive the car for the next cycle, which closes the loop.

One of the main goals for control engineers is to intelligently design the control law for C to progressively minimize e by computing a set of acceleration values, such that the system output will converge to the desired reference. PID is the most common control law used in practice, and used in our example. It corrects the next input based on proportional (K_p), integral (K_i), and derivative (K_d) gains w.r.t. the error.

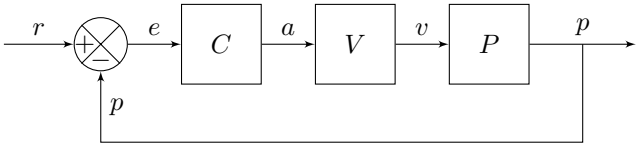

Figure 1: Block diagram of the car position tracking example

Next, the performance of the design needs to be evaluated, e.g., by simulation, model checking, or theorem proving.

Finally, if the evaluation is undesirable, the designer needs to redesign the hybrid system and then re-evaluate the design. Otherwise, we can build a prototype for trial.

In this work, we propose a static checker based on the Laplace transform. It aims to detect performance or safety requirement violations just after a design

[1]The example is borrowed and adapted from [22].

is finished or partially finished. This prevents users from performing expensive design evaluation tasks until those violations are reviewed or fixed.

For example, let us consider the design of Fig. 1 against the safety requirement $\forall t \cdot p(t) < r$. The relationship between the input reference r and the output car position p can be transformed into a transfer function $\frac{K_d s + K_p}{s^2 + K_d s + K_p}$ by the Laplace transform. We assume that this transfer function is partially defined, i.e., it is parameterized by the symbolic values K_p, K_i and K_d with constraints $K_p \leq 2$, $K_i = 0$, $K_d > 0$ and $5K_d < K_p$; and is approximated by $\frac{K_p}{s^2 + K_d s + K_p}$ using the technique discussed in Section 4.5. Static checking the approximated transfer function returns `sat` to indicate requirement violation. Our checker also returns a counterexample where the system, under $K_p = 0.25, K_d = 0.03$, and excited by a step input $r = 1$, produces an output trajectory p whose max value is about 1.67 (that is outside the safety range since $not(p(25.33) = 1.67 < 1 = r)$) at time $t = 25.33$. The users can use provided counterexample to confirm that the approximation preserves the significant dynamics of the original transfer function, and to reproduce the requirement violations in the original design by plotting or simulation.

3 Background

3.1 Laplace transform

The Laplace transform ($\mathcal{L}$) of a time-domain function $f(t)$ is defined by $\mathcal{L}\{f(t)\} = \int_0^\infty f(t)e^{-st}\,dt$, which results a function in the s-domain (where s is in the complex plane). A function $f(t)$ can be Laplace transformed if it is of exponential order. Exponential order means that there exists a real number σ such that $\lim_{t\to\infty} |f(t)e^{-\sigma t}| = 0$. The decaying exponential term $e^{-\sigma t}$ in the integrand ensures convergence. This means that even if $f(t)$ does not vanish as t goes to infinity, the integrand will still vanish since the value of σ ensure the exponential term grows at a faster rate than f.

Properties and rules of the Laplace transform can be derived from its definition (Table 2 in Appendix A gives a set of them that are used in this work). When designing hybrid systems, it is often helpful to obtain a **transfer function** by rearranging the Laplace transform result into a ratio of output to input. For example, the V component in the car position tracking example corresponds to a transfer function $\frac{V(s)}{A(s)} = \frac{1}{s}$ (see Appendix A for its derivation). The convolution rule of the Laplace transform is especially useful which allows composing transfer functions algebraically. For example, as $\frac{V(s)}{A(s)} = \frac{1}{s}$, and $\frac{P(s)}{V(s)} = \frac{1}{s}$, we can compose them algebraically using the convolution rule to obtain a new transfer function $\frac{P(s)}{A(s)} = \frac{1}{s^2}$, which directly shows how the acceleration affects the position.

3.2 Inverse Laplace transform

The inverse Laplace transform ($\mathcal{L}^{-1}$) of a function $F(s)$ is defined by $\mathcal{L}^{-1}\{F(s)\} = \frac{1}{2\pi j}\int_{\sigma-j\infty}^{\sigma+j\infty} F(s)e^{+st}\,dt$, which results a time-domain function $f(t)$. By the Lerch's theorem [6], if a function $F(s)$ has the inverse Laplace transform $f(t)$, then $f(t)$ is uniquely determined.

A standard problem in control is to understand the behaviors of the system output in the time-domain, which can be solved through the inverse Laplace transform: let $Y(s) = H(s)U(s)$, where $U(s)$ and $Y(s)$ represents the system input and output respectively in the s-domain, and $H(s)$ is the transfer function to be designed that connects the two. By design $H(s)$ and fixing $U(s)$, we can compute $Y(s)$ by convolution. Then, by inverse Laplace transforming $Y(s)$, we can study its corresponding function $y(t)$ to understand the behaviors of the system output in the time-domain.

As $Y(s)$ might be complex that is difficult to be directly inverse Laplace transformed, engineers usually break up $Y(s)$ by partial fraction expansion: any $Y(s)$ can be represented by a rational function in the form of $\frac{b_1 s^m + b_2 s^{m-1} + \ldots + b_{m+1}}{s^n + a_1 s^{n-1} + \ldots + a_n}$. We consider proper functions in this work where $m \leq n$ and further work should be carried out for other functions. By factoring the polynomials, this same function can be also expressed in terms of the product of factors as $Y(s) = K\frac{\Pi_{i=1}^{m}(s-z_i)}{\Pi_{i=1}^{n}(s-p_i)}$ [13]. Where $s = z_i$, s is referred to as a **zero** of the function; and where $s = p_i$, s is referred to as a **pole** of the function. By partial fraction expansion, we can rearrange $Y(s) = \sum_{i=1}^{n}\frac{C_i}{s-p_i}$, where C_i are coefficients that can be determined by the cover-up method [13]. Thus, by inverse Laplace transforming $Y(s)$, we have $y(t) = \sum_{i=1}^{n} C_i e^{p_i t}$ as its time-domain correspondence.

Notice that complex poles always come in pairs in the form of $p = \alpha + \beta j$ and $p^* = \alpha - \beta j$ such as $Y(s) = \ldots + \frac{C_1}{s-p} + \frac{C_1^*}{s-p^*}$. In this case, inverse Laplace transforming $Y(s)$ results[2]: $y(t) = \ldots + 2|C_1|e^{\alpha t}cos[\beta t + arg(C_1)]$.

3.3 Stability and final value theorem

A function $Y(s)$ that represents the system output is stable if all its poles have negative real parts in the complex plane, and is unstable otherwise.

As we see from Section 3.2, the real part of each pole would dictate whether its corresponding exponential term in $y(t)$ is decaying or growing. Thus, if all poles have the negative real part, $y(t)$ will have a steady state (where the system state is unchanging in time).

Another especially useful property of the Laplace transform known as the final value theorem allows us to compute the constant steady state value of $y(t)$ given its Laplace transform $Y(s)$. The final value theorem states that if all poles of $sY(s)$ are in the left half of the complex plane, then $\lim_{t\to\infty} f(t) = \lim_{s\to 0} sY(s)$. Notice that the final value theorem only applies to stable systems (as its restrictions on the positions of the poles in the complex plane), since the

[2] $arg(C_1)$ denotes the argument of C_1.

states of unstable systems cannot converge to a steady state and would not have a final value.

3.4 Order of the transfer functions

Given a transfer function $H(s)$ in the form of $\frac{b_1 s^m + b_2 s^{m-1} + ... + b_{m+1}}{s^n + a_1 s^{n-1} + ... + a_n}$, the highest exponent n is called the order of $H(s)$. In this work, we mainly consider the standard 2nd-order transfer functions, where $H(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$, ζ is called the **damping ratio** and ω_n is called the **un-damped natural frequency**. The reason we consider the transfer functions in the standard 2nd-order form is that ζ and ω_n tell a lot about the performance and characteristics of $H(s)$. In the next section, we discuss how to exploit ζ and ω_n for estimating the time-domain metrics. Moreover, we will discuss how to manage more complex transfer functions in Section 4.5.

4 Time-domain metrics estimation

In this work, we are interested in 5 types of time-domain metrics: overshoot, peak time, rise time, settling time and DC gain. The definitions, properties, and estimations of these time-domain metrics give us a way to formulate proof obligations to static check reference tracking system designs against performance or safety requirements.

To simplify our discussion, let:

- $H(s)$ (H for short) be a standard 2nd-order transfer function ($\frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$) for the designed system, where ζ denotes its damping ratio, and ω_n is its un-damped natural frequency, and
- $U(s)$ (U) be the system input. We will fix the input signal to be a step input for 2 reasons: 1) the considered time-domain metrics have a clear definition for step inputs. 2) to demonstrate how to derive the estimations for the considered time-domain metrics. In addition, we symbolically specify that the amplitude of the step input is A (i.e., $U = \frac{A}{s}$), and
- $Y(s)$ (Y) be the system output, where $Y = HU$, and
- Y and H be stable, as the considered time-domain metrics only make sense for stable systems, and
- $y(t)$ (y) be the inverse Laplace transform of Y in the time-domain, and of type $\mathbb{R}^+ \to \mathbb{R}$.

4.1 DC gain

Definition and property. The DC gain is the ratio of the system's steady state to the system input.

We use a predicate $input(y, A)$ to denote that the system step input has an amplitude of A. We use a predicate $dc(y, M_{dc})$ to denote that the DC gain of y is M_{dc}, and a predicate $steady(y, V_{dc})$ to denote that the steady state value of y is V_{dc}.

Then, we define our first domain-specific property P_{dc} for the DC gain:

$$P_{dc}(y) \hat{=} \forall A, M_{dc} \cdot input(y, A) \wedge dc(y, M_{dc}) \rightarrow steady(y, M_{dc}A)$$

which allows the derivation of steady state value from the (given) input and the (estimated) DC gain.

Estimation. The DC gain is estimated by applying the final value theorem:

$$E_{dc}(y, A) \hat{=} dc(y, \frac{\lim_{s \to 0} sY(s)}{A}) = dc(y, \lim_{s \to 0} H(s))$$

4.2 Overshoot and peak time

Definitions and properties. Overshoot is the maximal amount that y exceeds its steady state v divided by v. Peak time is the first time that takes the system output y to reach the overshoot point.

We use a predicate $overshoot(y, M_p)$ to denote that y has the overshoot M_p, and use a predicate $peak(y, t_p)$ to denote that the peak time of y is t_p.

Then, we define a domain-specific property P_{Mp}:

$$P_{Mp}(y) \hat{=} \forall M_p, t_p, v \cdot overshoot(y, M_p) \wedge peak(y, t_p) \wedge steady(y, v) \rightarrow M_p = \frac{y(t_p) - v}{v}$$

which is the definition of the overshoot. It also allows the derivation of overshoot point value from the (estimated) overshoot and the steady state value.

The domain-specific property P^1_{tp} states that the overshoot point is the maximum value of y:

$$P^1_{tp}(y) \hat{=} \forall t_p \cdot (peak(y, t_p) \rightarrow max(y, t_p))$$

where it uses an auxiliary predicate $max(y, t_x)$ that defines as follows:

$$max(y, t_x) \hat{=} (\forall t \cdot y(t) \leq y(t_x))$$

The domain-specific property P^2_{tp} states that the peak time is counted by the first time that y reaches its overshoot point:

$$P^2_{tp}(y) \hat{=} \forall t_p \cdot (peak(y, t_p) \rightarrow first(y, t_p))$$

where it uses an auxiliary predicate $first(y, t_x)$ that defines as follows:

$$first(y, t_x) \hat{=} (\forall t \cdot y(t) = y(t_x) \rightarrow t \geq t_x)$$

P^2_{tp} implies that the occurrence for the overshoot point of y before the peak time is unique.

Estimations. The damping ratio ζ of H plays an important role in estimating overshoot and peak time of Y. It can vary from un-damped ($\zeta = 0$), under-damped ($\zeta < 1$) through critically-damped ($\zeta = 1$) to over-damped ($\zeta > 1$).

For H that is un- or under-damped ($0 \leq \zeta < 1$), the discriminant of the quadratic equation $s^2 + 2\zeta\omega_n s + \omega_n^2$ will be less than 0, Thus, H will have two complex poles, which results sinusoidal behavior in its time-domain function (Section 3.2). It is this sinusoidal behavior being propagated from H to Y, and results in the overshoot of y.

Specifically, y found from the inverse Laplace transform of Y is:

$$y = A - Ae^{-\sigma t}\sqrt{1 + \frac{\sigma^2}{\omega_d^2}}\Big(cos(\omega_d t) + \frac{\sigma}{\omega_d} sin(\omega_d t)\Big)$$

where $\omega_d = \omega_n\sqrt{1-\zeta^2}$ and $\sigma = \zeta\omega_n$.

When, y reaches its maximum value, its derivative will be zero:

$$\dot{y} = e^{-\sigma t}\Big(\frac{\sigma^2}{\omega_d^2} sin(\omega_d t) + \omega_d sin(\omega_d t)\Big) = 0$$

This occurs when $sin(\omega_d t) = 0$. Thus, the peak time, i.e., the first time to reach the overshoot point, needs to satisfy $\omega_d t_p = \pi$. Consequently, we can estimate peak time by [13]:

$$E_{tp}(y, \zeta, \omega_n) \mathrel{\hat{=}} 0 \leq \zeta < 1 \rightarrow peak(y, \frac{\pi}{\omega_d})$$

Using the estimated value of peak time t_p into the time-domain function y, one can derive the estimation for the overshoot [13]:

$$E^1_{Mp}(y, \zeta) \mathrel{\hat{=}} 0 \leq \zeta < 1 \rightarrow overshoot(y, e^{\frac{-\pi\zeta}{\sqrt{1-\zeta^2}}})$$

When ζ is given numerically, E^1_{Mp} can be computed easily. However, when it is represented from symbolic values, solving E^1_{Mp} in the SMT solver involves non-linear non-polynomial real arithmetic, which often results in inconclusive results. Therefore, we replace E^1_{Mp} with the following abstract estimation when ζ is symbolically represented:

$$E^3_{Mp}(y, \zeta, c) \mathrel{\hat{=}} 0 \leq \zeta < 1 \rightarrow overshoot(y, c) \wedge 0 < c \leq 1$$

For H that is critically- or over-damped, the discriminant of the quadratic equation $s^2 + 2\zeta\omega_n s + \omega_n^2$ will be greater or equals to 0. Thus, H only has real poles, which does not result in sinusoidal behavior and overshoot:

$$E^2_{Mp}(y, \zeta) \mathrel{\hat{=}} \zeta \geq 1 \rightarrow overshoot(y, 0)$$

4.3 Rise time

Definition and property. Rise time is the time that takes the system y from the set point a to reach a new set point b for the first time.

We use a predicate $rise(y, a, b, t_r)$ to denote that y has the rise time t_r which takes from the set point a to b for the first time. Then, we have the following property:

$$P_{tr}(y) \hat{=} \forall t_r, a, b \cdot rise(y, a, b, t_r) \rightarrow \\ (\exists t_a \cdot y(t_a) = a \wedge first(y, t_a) \wedge y(t_a + t_r) = b \wedge first(y, t_a + t_r))$$

That is the predicate $rise(y, a, b, t_r)$ implies that there exists t_a such that y take t_a to first reach the set point a, and after t_r it first reaches the set point b.
Estimation. Although it is difficult to obtain exact analytic expressions for the rise time, a linear approximation have been used to estimate rise time for un-damped or under-damped systems [9]:

$$E_{tr}(y, \zeta, \omega_n) \hat{=} \forall r \cdot input(y, r) \wedge 0 \leq \zeta < 1 \rightarrow rise(y, 0.1r, 0.9r, \frac{2.16\zeta + 0.60}{\omega_n})$$

Notice that the expression $\frac{2.16\zeta+0.60}{\omega_n}$ is used to exclusively estimate the rise time that takes from the set point of 10% of the input to reach 90% of the input. In addition, this estimation is a close approximation only for $0.3 \leq \zeta \leq 0.8$ [9].

4.4 Settling time

Definition and property. The settling time is the time taken y to reach and remain within a certain tolerance δ of its steady state value.

We use a predicate $settle(y, \delta, t_s)$ to denote that y takes the settling time of t_s to reach and remain within δ of its steady state value. Then, we have the following property:

$$P_{ts}(y) \hat{=} \forall \delta, t_s, v \cdot steady(y, v) \wedge settle(y, \delta, t_s) \rightarrow (\forall t \cdot t \geq t_s \rightarrow |y(t) - v| \leq \delta v)$$

Estimation. To determine the time t_s for which the system output y remains within δ of the steady state value, the following approximation can be used [13]: $e^{-\zeta\omega_n t_s} < \delta$. Therefore, we have the following estimation for t_s:

$$E_{ts}(y, \delta) \hat{=} settle(y, \delta, -\frac{ln(\delta)}{\zeta\omega_n})$$

4.5 The simplification of higher-order transfer functions

Here, we introduce 2 simple techniques that can reduce the order of a given transfer function, while trying to maintain its significant dynamics in the time-domain. Recall from Section 3.2, any proper functions can be expressed as $F(s) = \sum_{i=1}^{n} \frac{C_i}{s-p_i}$ by partial fraction expansion, whose corresponding time-domain function is $f(t) = \sum_{i=1}^{n} C_i e^{p_i t}$.

For a stable function $F(s)$, the real part for each of its poles p_i controls how fast the term $e^{p_i t}$ in $f(t)$ reaches to its steady state. In other words, if the real part of p_i is far away from 0, $e^{p_i t}$ will reach its steady state very quickly, and would not contribute to $f(t)$ very much over time, and vice versa. This intuition leads to the first approximation technique: **to keep only dominant poles of a given transfer function whose real parts are close to** 0.

Similarly, each coefficient C_i in $F(s)$ controls the weight of $e^{p_i t}$ in $f(t)$. In other words, if C_i is small, $e^{p_i t}$ will not contribute to $f(t)$ very much over time, and vice versa. This intuition leads to the second approximation technique: **to keep only significantly weighted poles of a given transfer function**.

In summary, it is beneficial to study higher-order transfer functions by using lower-order approximated ones. However, we need approximations that can preserve the main characteristics of the original transfer function to make the study more meaningful. When we use techniques discussed above to obtain these approximations, it would depend on the user to decide a threshold for p_i or C_i to drop insignificant dynamics. In general, if the approximation precision is in doubt, the user could plot or simulate the approximated and the original transfer function to determine whether it is acceptable.

In addition to the simple approximation techniques, there are other approaches, e.g., a more sophisticated technique attempts to match the frequency response of the reduced-order transfer function with the original transfer function frequency response as closely as possible. Due to space, we refer [9] for more details of this approach.

5 Evaluations

5.1 Implementation

We have implemented our static checker as a `Python` program. Its inputs are: 1) the damping ratio ζ, and un-damped natural frequency ω_n for the 2nd-order transfer function of a given design. If the design is not in standard 2nd-order form, users need to approximate it using techniques discussed in Section 4.5. 2) additional user-specified constraints (if there are any). 3) desired safety or performance requirements.

Then, our static checker uses the inputs to formulate a PO according to the schema of $C \wedge E \wedge P \rightarrow S$:

- C represents additional user-specified constraints, e.g., types/values of system states and design parameters.
- E and P represent estimations and properties of time-domain metrics for the given system respectively. Notice that when ζ and ω_n are given by numerical constants, our checker performs numerical static checking by using these constants to compute time-domain metrics. When ζ and ω_n are given symbolically, our checker represents time-domain metrics estimations as symbolic formulas to perform symbolic static checking.

- S specifies desired safety or performance requirements.

Finally, we check the validity of the PO (i.e., $\neg$ PO) using the `Z3` SMT solver (version 4.8.12). The checker returns `sat` and a counterexample when requirement violations are detected, or `unsat` if our checker does not find requirement violations, or `unknown` if the checker cannot solve.

On conclusive checking result (`sat` or `unsat`), the user should first determine whether the approximation preserves the significant dynamics of the original design (e.g., assisted by the counterexample found by our static checker to plot the approximated and original transfer functions against the input). If not, the static checking result is less meaningful, and the user could try to re-approximate. Otherwise, the user can review/fix the design based on the static checking result and then decide whether to perform more expensive design evaluation tasks next. Our implementation is publicly available at: `http://github.com/veriatl/checkhybrid/`.

5.2 Research questions and evaluation setup

We formulate 2 research questions to evaluate the correctness and efficiency of our static checker:

1. Can our checker identify the requirement violations in a given design? If not, what are the causes?

2. Can our checker effectively provide useful information for the user to reproduce the bugs in the design?

To answer our research questions, we evaluate our checker against 10 reference tracking systems from the literature and textbooks [22, 9]. The evaluation is performed on an Intel i7-10510u machine with 16 GB of memory running the Linux operating system. We refer to our online repository for the complete artifacts used in our evaluation.

5.3 Evaluation results

Our evaluation result is summarized in Table 1. Each system is identified by a unique ID, followed by its original transfer function and the approximated one (not shown when the original transfer function is already in the standard 2nd-order form). Then, we list the type of the requirement specified on the design (SF and PF for safety and performance requirement respectively), and the type of static checking method (N and S for numerical and symbolic static checking respectively). The last column lists the actual checking result from the SMT solver for the approximated design, and the expected one on the original design (in brackets, omitted when it is the same as the actual one).

We report that our checker return results in around 1s for all the evaluated cases. It thus allows users to have immediate feedback on the design. Among the 5 cases of numerical static checking, all of them automatically return conclusive

Table 1: Evaluation results for static checking of hybrid system designs

ID.	Org.	Approx.	Req.	Method	Actual(Expect)
ex.nmp	$\frac{-0.5s+1}{s^2+3s+1}$	$\frac{1}{s^2+3s+1}$	SF	N	unsat(sat)
ex.dp0503	$\frac{K}{s^2+qs+K}$	-	PF	S	sat
ex.dp0504	$\frac{10K}{(s+3)(s+7)(s+70)+10K}$	$\frac{K/7}{21+K/7}\,\frac{21+K/7}{(s+3)(s+7)+K/7}$	PF	N	sat
ex.dp0506	$\frac{K_1}{s^2+(K_1K_2+1)s+K_1}$	-	PF	N	unsat
ex.plane	$\frac{114K_p}{s^3+11.4s+14s+114K_p}$	$\frac{11.29K_p}{s^2+\sqrt{1.92-2.91K_p}s+11.29K_p}$	PF	S	sat
ex.car			SF	N	sat
ex.car2	$\frac{K_ds+K_p}{s^2+K_ds+K_p}$	$\frac{K_p}{s^2+K_ds+K_p}$	SF	N	unsat
ex.car3			SF	S	sat
ex.ap0506	$\frac{KK_m}{s^2+(K_mK_b+0.01)s+KK_m}$	-	PF	S	sat
ex.ap0509	$\frac{K_ps+K_i}{s^4+40s^3+375s^2+K_ps+K_i}$	$\frac{\omega_n^2}{(s^2+\sqrt{2}\omega_ns+\omega_n^2)}$	PF	S	sat

results, and 4 of them return the correct result by manual inspection. In these cases, if a counterexample is returned, it will contain estimated time-domain metrics on the approximated design, which is shown to be a representative indicator for requirement violation of the original design (we refer to Appendix B.1 for an example). This level of efficiency is mainly because when ζ and ω_n are given by numerical values, our checker uses these values to compute time-domain metrics to constants, and propagate them via predicates as given in Sections 4.1-4.4. This avoids nonlinear real arithmetic SMT solving. We also simplify our encoding by reducing uninterpreted functions and eliminating quantifiers to help SMT solver return conclusive results. The case *ex.nmp* returns an incorrect result due to an unsound approximation. By examining, the original design has a positive zero in the nominator (a.k.a., non-minimum phase zero). It delays the system, by first going in the opposite direction of the desired reference. However, our approximation incorrectly drops this behavior when using the transfer function reduction techniques in Section 4.5, which prevents our checker to detect the requirement violation.

We have 5 cases that perform symbolic static checking. They all automatically return conclusive and correct checking results. We also confirm that the

generated counterexample helps us to simulate the original transfer function to reproduce the requirement violations. The efficiency of symbolic static checking is because we replace complex analytical estimations with simple abstract ones (we refer to Appendix B.2 for an example). However, as the abstract estimations lose the exact formula to calculate time-domain metrics, our symbolic checking is currently accompanied by an automated calibrating process, i.e., using the counterexample generated by the symbolic checking to perform another round of numeric checking to produce a more accurate counterexample.

5.4 Discussion

In summary, our experiment shows that our checker can identify the requirement violations in a given hybrid systems design provided that the approximation preserves the significant dynamics of the original design (research question 1). It can also automatically provide static checking result and counterexample for the user to reproduce bugs in the design (research question 2). Due to non-linear non-polynomial real arithmetic involved in the time-domain metrics estimation, we abstract estimation formulas with boundary conditions to return conclusive checking result. Then, we implement an automated calibration process to compensate for the loss of precision in the counterexample induced by this abstraction. Our experiment also shows us more insight about limitations and improvement opportunities about our approach, which we will discuss next.

Usability. As described in Section 5.1, the overhead to apply our approach resides in transforming a design into a 2nd-order transfer function. We use existing APIs in Python/Matlab: a) to prepare the transfer function from a given design (e.g., `feedback`, `series` for transfer function construction). b) to approximate a transfer function (e.g., `residue` for identifying candidate terms to remove). c) to ensure an approximation preserves the significant dynamics of the original design (e.g., `plot` for graphical comparison). We find that this overhead is relatively low compared to other evaluation tasks (e.g., iterative test selection for simulation [2, 8, 11], or refining unknown parameters to reduce the search space for model checking [14, 20, 17], or synthesizing and deductive proving Lyapunov function for theorem proving [1, 26, 22]). Therefore, we are convinced that performing a lightweight static checking would prevent users from performing more expensive evaluation tasks until reported violations are reviewed or fixed. Moreover, users would gain more confidence to develop hybrid systems based on the checked systems. To make our checker easier to use, we plan to automate the transfer function approximation for the user and propose convenient ways to introduce auxiliary predicates that quantify truncated transfer functions in the static checking process.

Soundness and completeness. There are potentially 3 types of approximation errors in the described estimation methods: 1) When approximate higher-order transfer function into a lower-order one, we try to truncate insignificant dynamics. 2) When we estimate time-domain metrics, we use the formula given in [13, 9]. 3) When we encode and compute time-domain metrics, we approximate transcendental numbers (e.g., e or π) in the formula by numerical roundup.

While these approximation errors introduce unsoundness and incompleteness in our approach, the goal for the proposed static checker is to quickly and statically analyze for obvious requirement violations without much effort. This is to prevent users from performing more expensive evaluation tasks until reported violations are reviewed or fixed. To improve the soundness and completeness of our analysis, we plan to improve the approximation precision, e.g., by finding more accurate analytical estimation methods for time-domain metrics.

Applicability. The demonstrated approach is designed for linear reference tracking systems. In this context, it is reasonable to: 1) represent the input as a step input (which characterizes a sudden change of reference), 2) represent the controlled system as a 2nd-order transfer function (which characterizes a low-pass filter-like behavior), 3) consider controlled stable systems (the system itself does not need to be stable, but should be stable after control). 4) assume system dynamics under zero initial conditions (since it simplifies the analysis). We are interested in how to apply our approach in more contexts, e.g., tracking of a sinusoidal input. In addition, reference tracking systems are the simplest hybrid systems that do not exhibit any discrete jumps. However, their behaviors are crucial to reason the correctness of high-level complex hybrid systems that orchestra them. We plan to integrate our approach with other evaluation approaches (e.g. [5, 14, 21]) to reason the safety of more complex hybrid systems.

6 Two Static Checking Examples

B.1. Numerical static checking (*ex.car*)

Consider the design in Fig. 1, the system output is the car position, given by $Y(s)$, where $Y(s) = H(s)U(s)$. $H(s)$ is the designed closed-loop transfer function, and $U(s)$ is a step input for specifying the desired reference position $r = 1$. Let $y(t)$ represent the corresponding time-domain function of $Y(s)$. We want to know that under a PD control law of $K_p = 2, K_d = 0.14$, whether the safety property: $\forall t \cdot y(t) < r$ would hold for this design.

Under zero initial conditions, we calculate the transfer function $H(s)$ for the closed-loop system by applying the Laplace transform:

$$H(s) = \frac{K_d s + K_p}{s^2 + K_d s + K_p} = \frac{K_d}{K_p} \frac{K_p s}{s^2 + K_d s + K_p} + \frac{K_p}{s^2 + K_d s + K_p}$$

In this case, $H(s)$ can be divided into two sub transfer functions by partial fraction expansion, with weight $\frac{K_d}{K_p}$ and 1 respectively. As suggested by the chosen constant values in the PD control law, $\frac{K_d}{K_p}$ is less than one tenth of 1. Thus, we decide to approximate $H(s)$ by $\hat{H}(s)$:

$$\hat{H}(s) = \frac{K_p}{s^2 + K_d s + K_p}$$

where $\hat{\zeta} = \frac{K_d}{2\sqrt{K_p}} = 0.05$, and $\hat{\omega_n} = \sqrt{K_p} = 1.41$[3]. Thus, we have $\hat{Y}(s) = \hat{H}(s)U(s)$ to be an approximation of $Y(s)$.

Next, we apply our estimations for time-domain metrics on $\hat{Y}(s)$. Combined with domain-specific properties of time-domain metrics shown in Section 4, we generate a PO shown in Fig. 2 (to save space, only formulas that are related to the concerned safety property are shown). We have:

$$
\begin{aligned}
C &\triangleq (\hat{y} : R^+ \to R) \wedge (r : R^+) \wedge r = 1 \wedge input(\hat{y}, r) \wedge \\
&\quad \hat{\zeta} = 0.05 \wedge \hat{\omega_n} = 1.41 \wedge ... \\
E &\triangleq dc(\hat{y}, 1) \wedge overshoot(\hat{y}, 0.86) \wedge peak(\hat{y}, 2.22) \wedge ... \\
P &\triangleq P_{dc} \wedge P_{Mp} \wedge P^1_{tp} \wedge ... \\
S &\triangleq \forall t \cdot \hat{y}(t) < r \\
PO &\triangleq \neg(C \wedge E \wedge P \to S)
\end{aligned}
$$

Figure 2: Proof obligation generation for hybrid system design of Fig. 1

- C encodes the system context. Here, $\hat{y}$ is the corresponding time-domain function of $\hat{Y}(s)$. We also have a step input $r = 1$, and numerical constants for $\hat{\zeta}$ and $\hat{\omega_n}$, and etc..
- E encodes the estimated time-domain metrics for $\hat{y}$, where its DC gain is 1 (by E_{dc}), and overshoot by 0.86 (by E^1_{Mp}), and time to the first peak is approximately 2.22 seconds (by E_{tp}), and etc..
- P encodes the properties of time-domain metrics. To verify the considered safety property, P_{dc} derives the steady state value of $\hat{y}$. P_{Mp} relates overshoot with peak time to derive the overshoot point value $\hat{y}(t_p) = 1.86r$. P^1_{tp} is used to derive $max(\hat{y}, t_p)$.
- S is the desired safety property, i.e., the car position never exceeds the reference position r.
- Finally, a PO is formulated and checked by the SMT solver.

Upon receiving the checker result, we first plot $H(s)$ and $\hat{H}(s)$ against a step input (Fig. 3). The plotting result confirms that the approximation preserves the significant dynamics of the original system. Thus, we should further examine the static checking result. Then, our checker returns `sat` which suggests a requirement violation. The produced counterexample shows that the time-domain function $\hat{y}$ has a maximum of 1.86 at $t = 2.22$ that violates the safety requirement. Guided by this information, we then confirm on the plot that this is also a safety requirement violation in the original design.

[3]Numbers are rounded to two decimal places for simplification

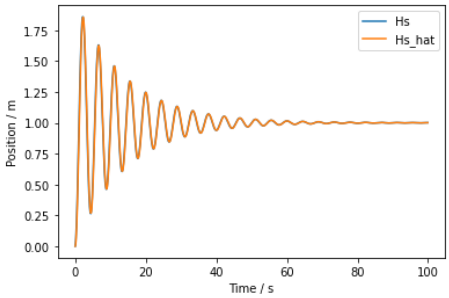

Figure 3: Simulation result for the approximated and original design of car position tracking system, under the reference step input $r = 1$, and the controller law of $K_p = 2$, $K_i = 0$ and $K_d = 0.14$

B.2. Symbolic static checking (*ex.plane*)

The example is adapted from [9], which controls the bank angle of an airplane by using the aileron actuator. Bank angle is the angle between the wings and the horizon.

An aileron forms part of the trailing edge of each wing of a fixed-wing aircraft. By actuating the aileron up or down, we can control the bank angle of the aircraft. The goal of the design is to develop a proportional controller that gradually actuates the aileron to maintain the bank angle at a desired constant ϕ_d. We define a requirement for the design such that the bank angle does not overshoot. The original design is given by a closed-loop transfer function:

$$H(s) = \frac{114K_p}{s^3 + 11.4s^2 + 14s + 114K_p}$$

Using an approximation technique that preserves similar frequency responses [9], we obtain a standard 2nd-order closed-loop transfer function:

$$\hat{H}(s) = \frac{11.29K_p}{s^2 + \sqrt{1.92 - 2.91K_p}s + 11.29K_p}$$

Therefore, exciting the system with a step input $U(s)$, we have $\hat{Y}(s) = \hat{H}(s)U(s)$ that represents the output of the approximated system.

The closed-loop transfer function $\hat{H}(s)$ is parameterized by a symbolic proportional gain K_p. Due to stability, we might have a restriction on K_p such that it needs to be strictly between 0 and 0.65. A typical question we might want to ask our static checker is that, under such restriction, is our desired requirement held? For that, a PO is formulated as shown in Fig. 4. The result of *PO* in the SMT solver is `sat`. By examining the calibrated counterexample, there is a $K_p = 0.14$, which is within the acceptable range for stability. We use this value to plot the approximated and the original transfer functions against the step input. The result shows that it is a close approximation of the original one (Fig. 5). Then, we check the counterexample and find that $\hat{y}$ has

$$
\begin{aligned}
C &\hat{=} 0 < K_p < 0.65 \wedge \hat{\zeta} = \sqrt{\frac{0.043}{K_p} - 0.065} \wedge \hat{\omega_n} = \sqrt{11.29 K_p} \wedge ... \\
E &\hat{=} E^1_{Mp} \wedge E^2_{Mp} \wedge ... \\
P &\hat{=} ... \\
S &\hat{=} overshoot(\hat{y}, 0) \\
PO &\hat{=} \neg(C \wedge E \wedge P \rightarrow S)
\end{aligned}
$$

Figure 4: Proof obligation generation for hybrid system design of bank angle control system

$overshoot = 0.16$ that violates the requirement S. We then confirm on the plot that this is also a requirement violation in the original design.

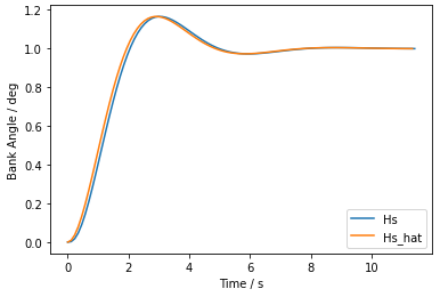

Figure 5: Simulation result for the approximated and original design of the bank angle control system, under the reference step input $r = 1$, and the controller law of $K_p = 0.14$

7 Related Works

Researchers have developed various techniques to evaluate hybrid system designs against their requirements.

Simulation. There are several commonly encountered engineering tools to simulate hybrid system designs, such as `MATLAB` (Simulink, Stateflow) and `Mathematica`. In general, it is not always possible to simulate all possible cases. Thus, one of the main challenges for researchers is to search representative inputs for simulation. Mature input searching tools around MATLAB exist [2, 8]. They use robustness as an optimization function to find inputs that steer the system towards violation of a given temporal logic requirement. However, their performance varies with the structure of the problem at hand and the chosen optimization algorithm. Ernst et al. develop an algorithm based on a probabilistically directed search that adapts to the difficulty of a problem at hand [11].

Model checking. Many tools are developed for bounded model checking.

Among these, HyTech is a model checker for linear hybrid system designs that focuses on exact computations of reachable states [16]. Such exact computation is possible for simple continuous dynamics by applying linear algebra, but easily becomes undecidable when dynamics switching is involved in the system modeling. SpaceEx takes another approach for linear systems by over-approximating the set of reachable states on a bounded time-interval, which could scale to high-dimensional systems [18]. Flow* is a tool that approximates nonlinear system dynamics using Taylor models and template polyhedra [4]. dReach is another bounded model checker designed to handle general hybrid systems with nonlinear differential equations and complex discrete mode-changes [17]. It is based on a δ-complete decision procedure for SMT formulas: δ-completeness allows a formula to be labeled satisfiable under user-specified numerical errors that can be tolerated in the analysis.

Theorem proving. KeYmaera X is a theorem prover for hybrid system designs based on differential logic (dL) [21, 15]. Special rules are designed to ease the complexity of deduction reasoning. In [22], Quesel et al. demonstrate how KeYmaera X and dL are used to verify a similar problem as given in Section 2. A global Lyapunov function is needed to construct an invariant to abstract system behaviors for safety verification. Hybrid Hoare logic (HHL) has been proposed by Liu et al. for a Hoare-style duration calculus based on hybrid communicating sequential processes [19]. Banach et al. propose a Hybrid Event-B language that applies software refinement methodology for developing hybrid systems and reasoning their safety [3]. The language introduces many user-friendly syntactic and semantic elements. Dupont et al. concretize the Hybrid Event-B language into the Event-B language [10]. In addition, they demonstrate how to encapsulate domain-specific knowledge of hybrid systems. Cheng and Méry try to improve the work of Dupont et al. by relaxing the constraints of Hybrid Event-B language to enable machine-checkable proofs and implementation-oriented refinement [5].

Our work is inspired by ESC/Java [12]. It is an extended static checker of Java. By introducing unsoundness and incompleteness at an appropriate level, it can achieve a better position in the coverage-to-effort design space. Similar to ESC/Java, although our analysis result is unsound and incomplete, it statically detects performance and safety requirement violations for linear reference tracking systems without much effort. It also prevents users from performing more expensive evaluation tasks until reported violations are reviewed or fixed. Moreover, users would gain more confidence to develop or evaluate hybrid systems based on the checked reference tracking systems. Thus, integrating our analysis in the traditional evaluation process of hybrid systems can be beneficial and complementary to existing techniques on simulation, model checking, and theorem proving.

The Laplace transform and its inversion provide a unique way to determine the time-domain function for the system state. It also has an algebraic nature to quantify complex designs as a whole, which provides an alternative view to cycle-based analysis such as in [21, 19, 3, 10, 5]. Moreover, it is an effective tool for robustness analysis. Thus, we are convinced that integrating more and more

domain knowledge associated with the Laplace transform will be quite useful for evaluating hybrid system designs. In addition to what we present in this work, Wang and Chen encode the Laplace transform and its properties in the `Coq` interactive theorem prover to verify the transformation correctness [28]. A similar effort has been found in `Isabelle` by Rashid and Hasan [25]. In previous versions, we have missed a few formalizations of the Laplace transform done in HOL Light theorem prover published in [27, 23, 24] and those works provide a general framework using theorem prover for analysing properties of Laplace Transform. These formalizations provide a rigorous basis for more analysis based on the Laplace transform.

8 Conclusion

In summary, we propose a static checker for checking reference tracking system designs against their performance and safety requirements. It aims to filter out designs that have obvious requirement violations. This is to prevent users from performing expensive design evaluation tasks until those violations are reviewed or fixed. In the process, our checker depends on domain knowledge of the Laplace transform to represent each design into a transfer function. Then, it estimates time-domain metrics and interprets the definitions and properties of time-domain metrics using first-order formulas over real numbers. This allows it to formulate a PO for checking requirement violations in the `Z3` SMT solver. Our checker can be applied to designs that are parameterized by either numerical or symbolic values. Our future work will focus on: 1) integrating with other evaluation approaches (e.g., [5, 18, 21]) to reason the safety of more complex hybrid systems. 2) reducing approximation errors in the static checking process.

References

[1] Ahmed, D., Peruffo, A., Abate, A.: Automated and sound synthesis of Lyapunov functions with SMT solvers. In: 26th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. vol. 12078, pp. 97–114. Springer, Dublin, Ireland (2020)

[2] Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In: 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 254–257. Springer, Saarbrücken, Germany (2011)

[3] Banach, R., Butler, M., Qin, S., Verma, N., Zhu, H.: Core Hybrid Event-B I: Single Hybrid Event-B machines. Science of Computer Programming 105, 92 – 123 (2015)

[4] Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: 25th International Conference on Computer Aided Verification. pp. 258–263. Springer, London, UK (2013)

[5] Cheng, Z., Méry, D.: A refinement strategy for hybrid system design with safety constraints. In: 10th International Conference on Model and Data Engineering. pp. 3–17. Springer, Tallinn, Estonia (2021)

[6] Davies, B.: Integral transforms and their applications, vol. 41. Springer Science & Business Media (2002)

[7] De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: 14th International conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer, Budapest, Hungary (2008)

[8] Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: 22nd International Conference on Computer Aided Verification. pp. 167–170. Springer, Edinburgh, UK (2010)

[9] Dorf, R.C., Bishop, R.H.: Modern control systems. Pearson Prentice Hall, 13 edn. (2016)

[10] Dupont, G., Ameur, Y.A., Pantel, M., Singh, N.K.: Handling refinement of continuous behaviors: A proof based approach with Event-B. In: 13th International Symposium on Theoretical Aspects of Software Engineering. pp. 9–16. IEEE, Guilin, China (2019)

[11] Ernst, G., Sedwards, S., Zhang, Z., Hasuo, I.: Fast falsification of hybrid systems using probabilistically adaptive input. In: International Conference on Quantitative Evaluation of Systems. pp. 165–181. Springer (2019)

[12] Flanagan, C., Leino, K.R.M., Lillibridge, M., Nelson, G., Saxe, J.B., Stata, R.: Extended static checking for Java. In: 2002 ACM SIGPLAN Conference on Programming language design and implementation. pp. 234–245 (2002)

[13] Franklin, G.F., Powell, J.D., Emami-Naeini, A., Powell, J.D.: Feedback control of dynamic systems. Prentice hall Upper Saddle River, NJ, 5 edn. (2006)

[14] Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: 23rd International Conference on Computer Aided Verification. pp. 379–395. Springer, Snowbird, UT, USA (2011)

[15] Fulton, N., Mitsch, S., Quesel, J., Völp, M., Platzer, A.: Keymaera X: an axiomatic tactical theorem prover for hybrid systems. In: 25th International Conference on Automated Deduction. pp. 527–538. Springer, Berlin, Germany (2015)

[16] Henzinger, T.A., Ho, P.H., Wong-Toi, H.: HyTech: A model checker for hybrid systems. International Journal on Software Tools for Technology Transfer 1(1-2), 110–122 (1997)

[17] Kong, S., Gao, S., Chen, W., Clarke, E.: dReach: δ-reachability analysis for hybrid systems. In: 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 200–205. Springer, London, UK (2015)

[18] Le Guernic, C., Girard, A.: Reachability analysis of linear systems using support functions. Nonlinear Analysis: Hybrid Systems 4(2), 250–262 (2010)

[19] Liu, J., Lv, J., Quan, Z., Zhan, N., Zhao, H., Zhou, C., Zou, L.: A calculus for hybrid CSP. In: 8th Asian Symposium on Programming Languages and Systems. pp. 1–15. Springer, Shanghai, China (2010)

[20] Minopoli, S., Frehse, G.: SL2SX Translator: From Simulink to SpaceEx Models. In: 19th International Conference on Hybrid Systems: Computation and Control. pp. 93–98. ACM, Vienna, Austria (2016)

[21] Platzer, A.: Logical Foundations of Cyber-Physical Systems. Springer (2018)

[22] Quesel, J.D., Mitsch, S., Loos, S., Aréchiga, N., Platzer, A.: How to model and prove hybrid systems with KeYmaera: a tutorial on safety. International Journal on Software Tools for Technology Transfer 18(1), 67–91 (2016)

[23] Rashid, A., Hasan, O.: Formal analysis of linear control systems using theorem proving. In: Duan, Z., Ong, L. (eds.) Formal Methods and Software Engineering. pp. 345–361. Springer International Publishing, Cham (2017)

[24] Rashid, A., Hasan, O.: Formalization of lerch's theorem using hol light. Jopurnzl of Applied Logic 5(8) (November 2018), https://www.collegepublications.co.uk/downloads/ifcolog00028.pdf

[25] Rashid, A., Hasan, O.: Formal analysis of continuous-time systems using Fourier transform. Journal of Symbolic Computation 90, 65–88 (2019)

[26] Tan, Y.K., Platzer, A.: Deductive stability proofs for ordinary differential equations. In: 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 181–199. Springer, Luxembourg, Luxembourg (2021)

[27] Taqdees, S.H., Hasan, O.: Formalization of laplace transform using the multivariable calculus theory of hol-light. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning. pp. 744–758. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

[28] Wang, Y., Chen, G.: Formalization of Laplace Transform in Coq. In: 4th International Conference on Dependable Systems and Their Applications. pp. 13–21. IEEE, Beijing, China (2017)

A Laplace transform rules

Table 2: The Laplace transform properties and rules used in this work

id	f(t)	F(s)	Comment
R_1	$\alpha f_1(t) + \beta f_2(t)$	$\alpha F_1(s) + \beta F_2(s)$	Superposition
R_2	$f(\alpha t)$	$\alpha F(s)$	Scaling
R_3	$f^m(t)$	$s^m F(s) - s^{m-1} f(0) - s^{m-2}\dot{f}(0) \ldots - f^{m-1}(0)$	differentiation
R_4	$\int_0^t f(x)\,dx$	$\frac{1}{s}F(s)$	integration
R_5	$f_1(t) * f_2(t)$	$F_1(s)F_2(s)$	convolution
R_6	1(t)	$\frac{1}{s}$	step
R_7	e^{-at}	$\frac{1}{s+a}$	exponent
R_8	$e^{-at}\sin bt$	$\frac{b}{(s+a)^2+b^2}$	

The Laplace transform properties and rules used in this work is shown in Table 2. To demonstrate its usage, considering the dynamics of the V component in the car position tracking example in Fig. 1. Its input is the acceleration a produced by the controller, and its output is the car velocity v. If we assume zero initial conditions of the system, the transfer function of V that defines its input/output relationship is:

$$\begin{aligned}
\mathcal{L}\{\dot{v} = a\} &\Leftrightarrow \mathcal{L}\{\dot{v}\} = \mathcal{L}\{a\} && \textbf{(Superposition)} \\
&\Leftrightarrow sV(s) + v(0) = A(s) && \textbf{(differentiation)} \\
&\Leftrightarrow sV(s) = A(s) && \textbf{(assumption)} \\
&\Leftrightarrow \frac{V(s)}{A(s)} = \frac{1}{s} && \textbf{(arrange)}
\end{aligned}$$