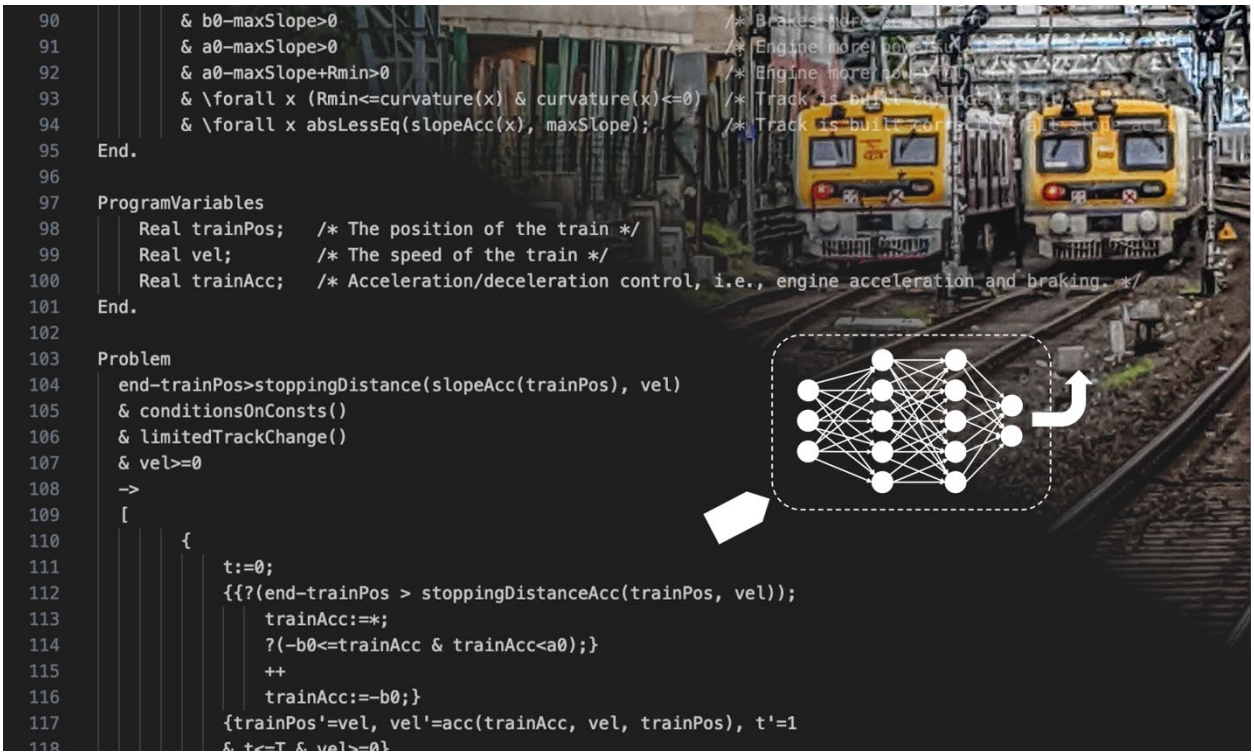
U.S. Department
of Transportation
Federal Railroad
Administration

Office of Research,
Development and Technology
Washington, DC 20590

Ensuring Safety in an AI-Enhanced PTC System



REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
25 August 2026	Technical Report	January 2021- December 2023

4. TITLE AND SUBTITLE

Ensuring Safety in an AI-Enhanced PTC System

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Aditi Kabra: ORCiD – 0000-0002-2252-0539
Stefan Mitsch: ORCiD – 0000-0002-3194-9759
André Platzer: ORCiD – 0000-0001-7238-5710
Christof Budnik: ORCiD – 0000-0002-5728-4134
Parinitha Nagaraja: ORCiD – 0009-0000-3900-8163

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
Subcontractor: Siemens Technology, Princeton, NJ 08540

8. PERFORMING ORGANIZATION REPORT NUMBER

If Any

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Department of Transportation
Federal Railroad Administration
1200 New Jersey Ave SE
Washington, DC 20590

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
DOT/FRA/ORD-26/18

12. DISTRIBUTION/AVAILABILITY STATEMENT
This document is available to the public through the FRA website.

13. SUPPLEMENTARY NOTES
COR: Jared Withers

14. ABSTRACT

Embedded software for train control is safety-critical because errors can have disastrous consequences. To ensure the safety of controllers, formal verification with computer-checked, repeatable mathematical proofs presents a particularly trustworthy method for controller design. The Federal Railroad Administration contracted a research team from Carnegie Mellon University to develop a provably safe, machine-learning based, predictive train control algorithm to control acceleration and braking along rail tracks. This research was conducted from January 2021 to December 2023. The team used formal verification as a tool to design and verify a symbolic train controller, and a testing-based approach to fill in appropriate values for the symbolic parameters of the formal model. The formalization addresses complex dynamics with transcendental arithmetic, competing forces with subtle interaction, and effects whose exact magnitude is unknown at proof time.

15. SUBJECT TERMS
Positive train control, formal verification, model validation, reinforcement learning

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**		85	**19b. TELEPHONE NUMBER** *(Include area code)*

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

Illustrations

Tables

Executive Summary

The Federal Railroad Administration (FRA) contracted a research team from Carnegie Mellon University to develop a provably safe, machine-learning based, predictive train control algorithm to control acceleration and braking along rail tracks. This research was conducted from January 2021 to December 2023. The team created a mathematically verified symbolic formal model to safeguard controller decisions, a testing-based approach to instantiate the symbolic formal model with concrete parameters that fit specific railroad and train operations, and used the verified safety envelope when training a machine-learning based controller. The benefit of this approach is that the verified formal model is fully symbolic; verified models can be customized with additional details of specific railroad companies and even additional details of specific trains, while inheriting the safety proof of the general symbolic model.

The team developed and verified a generic symbolic formal train model of train control and motion, based on constant, linear, and quadratic influences on speed. Researchers customized the generic symbolic formal train model to a verified symbolic freight train model, which estimates the constant, linear, and quadratic terms from surrogate parameters (e.g., train length, weight, aerodynamic and other resistive coefficients, bounds on locomotive tractive effort and brakes, and bounds on brake force provided by air brakes). To relate these proofs to realistic train operation, model instances were created from the symbolic freight train model using a testing-based validation method over a wide variety of train configurations. Model instances predicted a stopping distance from the verified symbolic predictions of the symbolic model. The testing-based parameterization of the surrogate parameters (e.g., train length) of the freight train model resulted in a stopping distance estimation that, on average, aligned well with the simulation.

Researchers created formally verified train controllers comprising a kinematics model with all its competing influences of track grade, curve resistance, air brakes, and Davis resistance. The team conducted testing-based parametrization using the Train Energy and Operations (TOES) simulator to obtain model instances of the formal freight train models that can predict (simulated) train behavior over a wide variety of train configurations. The team found that, on average, the parameters based on (Brosseau & Ede, 2009) and (Brosseau J. , Ede, Pate, Wiley, & Drapa, 2013) of the freight train model align well with the simulator. Techniques to couple formal models with reinforcement learning can be used to then turn formal models into reward functions to support the learning process itself and obtain agents that behave robustly inside the formally verified safety envelope.

1. Introduction

Embedded software for train control is safety-critical because errors can have disastrous consequences. To ensure the safety of controllers, formal verification with computer-checked, repeatable mathematical proofs presents a particularly trustworthy method for controller design. The Federal Railroad Administration (FRA) contracted a research team from Carnegie Mellon University to develop a provably safe, machine-learning based, predictive train control algorithm to control acceleration and braking along rail tracks. This research was conducted from January 2021 to December 2023. The team used formal verification as a tool to design and verify a symbolic train controller, and a testing-based approach to fill in appropriate values for the symbolic parameters of the formal model. The formalization addresses complex dynamics with transcendental arithmetic, competing forces with subtle interaction, and effects whose exact magnitude is unknown at proof time.

1.1 Background

Train controllers decide when to enforce braking to prevent movement authority (MA) violation and collisions. They must account for all the competing influences that govern train motion. Uphill slopes decrease velocity, for example, which decreases resistance, which permits a more rapid increase in velocity, slope, and curve effect, all while the train's brake force builds gradually until saturation as air pressure propagates along brake pipes. These complex interactions make it hard to design a safe and efficient train controller, and even harder to ensure it consistently remains safe. Researchers designed and verified train controllers for a freight train kinematics model (Brosseau J. , Ede, Pate, Wiley, & Drapa, 2013) (Brosseau & Ede, 2009), contributing a verified controller using generalizable, verified controller design techniques. These techniques can be extended and applied to produce verified controllers specialized for greater efficiency for a given railroad system or train category.

There have been many efforts to verify the safety of train control systems due to their significance as safety-critical transportation systems. One approach is extensive simulation of train braking models. However, simulation can only show safety in a limited number of cases and is less appropriate when free acceleration is interspersed with braking. Similar limitations apply to test-based safety assurance of train control. Therefore, in this study, the team used differential dynamic logic (dL), a logic with a deductive proof system for hybrid systems. Researchers developed mathematical proofs that guarantee safety over the infinite state space in a model of physical motion.

There have been many studies of formal verification in railway systems. Discrete aspects of train control have been verified at industrial scale. Many studies focus on scheduling trains to avoid route intersections. Train communication systems have been formally verified. Such studies are complementary to this study, which focuses on the motion of the train as it interacts with the environment.

Studies of train motion have verified the European train control system with moving blocks, and the Chinese train control system, while ignoring the effect of resistance, air brakes, track grade and curvature. The models of (Brosseau J. , Ede, Pate, Wiley, & Drapa, 2013) (Brosseau & Ede, 2009) have been verified while ignoring grade and curve. This study differs by accounting for all forces in (Brosseau J. , Ede, Pate, Wiley, & Drapa, 2013) (Brosseau & Ede, 2009), creating a controller designed and verified against realistic physics.

1.2 Objectives

The primary objective of this work is a formal verification results justifying the safety of train control. The secondary objective is the design of new, safe train controllers and their permissive set-valued safe control envelopes. Technical objectives are the contribution of ideas aiding effective, verified controller creation, like the techniques used by the formal proofs, and substituting a specialized ground physics model into controllers designed with an abstract mathematical model. A final objective is the testing-based estimation of model parameters through experimental validation of the verified controller and analysis of its behavior. To this end, the simulations in this study demonstrate the impact of verified train controllers in different terrain.

1.3 Overall Approach

Existing studies of formally verified train motion do not account for all effects (e.g., track grade, track curvature, resistance, and air brake propagation time), rendering their results inapplicable to most real-world scenarios. In this study, the team performs verification against the full dynamics of (Brosseau J. , Ede, Pate, Wiley, & Drapa, 2013) (Brosseau & Ede, 2009), in which these effects interact subtly with each other. The verification results are significant, because these parameters influence the motion of the train in safety-critical and/or performance-critical ways. Neglecting track slope profile and the gradual propagation of air pressure braking can render otherwise verifiably safe train controllers unsafe, since their influence may diminish the train's ability to decelerate, causing collisions.

Before verifying controller safety, the team first designed the controllers, balancing efficiency with provable safety. Conservative controllers are mathematically more simplistic and easier to design and verify, but are inefficient in railway operations, violating performance objectives. The team first presented a conservative safe controller and then demonstrated how to iteratively make it more efficient by exploiting characteristics of the physical train dynamics for safer control.

Train controllers are assessed relative to a (changing) destination stopping point called *end of authority (EOA)*. Overshooting the EOA is a safety violation because it risks collision with other trains. In this study, the team verified the absence of EOA overshoot when using the controllers in models (Brosseau J. , Ede, Pate, Wiley, & Drapa, 2013) (Brosseau & Ede, 2009).

To keep the proofs as general and widely applicable as possible, the team leveraged nondeterministic controllers and a paradigm of mathematical abstraction. Each controller was intentionally built to be set-valued so that *all* its control choices are simultaneously proved safe under all circumstances in the kinematics model. The safety of these controllers implies the safety of *all* their specializations, allowing railroads significant freedom in how to adapt the verified controllers for their purposes. Controller verification follows a two-stage process: first mathematical models of abstract train control motion are proved, and then proofs of the actual physical models of train control are obtained by *uniform substitution* to replace the abstract function symbols of the mathematical models with physical terms specific to the models (Brosseau J. , Ede, Pate, Wiley, & Drapa, 2013) (Brosseau & Ede, 2009) or even specific railroads.

Crucially, this approach uses three different types of models: (i) the high-fidelity physics model describing the kinematic motion of trains along the track, (ii) the generalized mathematical abstractions of the physics model developed in this study, and (iii) the simplified but *computable*

approximations of motion models used by the respective train controllers. The verification results prove that the safety of (i) derives from the safety of (ii) and that all control decisions following (iii) are safe in (ii).

The proof is written in differential dynamic logic, and performed using the hybrid systems theorem prover KeYmaera X. The team compared the efficiency with concrete control algorithms for several train consists (i.e., the arrangement of locomotives and cars) and scenarios. The verified models act as safety envelopes for unverified controllers via runtime enforcement checks using ModelPlex, and their behavior is illustrated in simulation. The proved models are permissive and only interfere in train control operation when acting otherwise risks MA violation.

The team then used a testing-based approach to parameterize the symbolic models from simulations. Researchers demonstrated the testing-based approach using the Train Energy and Operations (TOES) simulator. The experiments with TOES indicate that the testing-based parameterization of the freight train model results in stopping distance estimation that, on average, aligns well with the simulator.

In a final implementation step, the team combined the parameterized formal model and its verified safety envelope with reinforcement learning. With reinforcement learning, the team trained a machine learning agent (which chooses actions such as braking or coasting) to make appropriate decisions. The formal model and its proof indicate whether an action is safe in a certain situation. For their combination, the team viewed the verified safety envelope in a quantitative way; the safety envelope not only identifies whether the choices of a learned controller are safe or unsafe, but also measures the robustness of the choice (i.e., *how* safe or unsafe). The quantitative measure then indicates how well the chosen action performs in terms of the verified safety envelope so that the learning agent over many simulations improves its decision-making.

1.4 Scope

This report describes air brake propagation time (Section 4.4), which makes the verified controller most suitable for freight trains, where air brakes play a primary role in braking. Subsequently, the final proved model applies all the proof techniques from each iteration, resulting in a controller that retains the efficiency gains from each technique. It handles the interaction of air brakes, resistance approximation, and slope and curve estimation, which have non-trivial, interdependent interaction. The proof technique to handle Davis resistance is broadly applicable to other quadratic influences on train motion. The team extensively validated the new controller. A concise version of the reported work focusing on verification techniques was published separately in peer-reviewed literature (Kabra, Mitsch, & Platzer 2022) (Quin & Mitsch, 2023). This report, however, focuses on the application of the techniques to railroad systems and contains a more extensive evaluation.

1.5 Organization of the Report

Section 2 introduces the technical background for expressing and verifying formal models of train operation. Section 3 gives an overview of the formalization activities, building from basic models to a comprehensive model that considers air brakes and terrain. Section 4 discusses modeling and verification techniques to improve the efficiency of basic models without sacrificing their safety. Section 5 presents a testing-based approach to model parametrization.

Section 6 presents an evaluation of simulation architecture for reinforcement learning. Section 7 discusses control reinforcement learning from formal models. Section 8 discusses the moving block controller model, and Section 9 discusses moving block controller learning. Section 10 presents the conclusions made from this research.

2. Technical Background

This section describes the logic and train kinematics background necessary to the rest of the report. It contains a brief introduction to Differential Dynamic Logic (dL), in which the verified controllers are written and proved. Additionally, there is a brief recap the models from (Brosseau, et al., 2013; Brosseau & Ede, 2009) for describing the forces acting on a train.

2.1 Differential Dynamic Logic

dL is a logic with a deductive proof system for hybrid systems (Platzer, 2008; Platzer, 2017). Only a short overview is given here. More details and elaborate explanations of dL as a specification and verification language can be found in the literature (Platzer, Logical Foundations of Cyber-Physical Systems, 2018).

Differential dynamic logic extends first-order logic with the notion of hybrid programs. A hybrid program runs according to a binary relation between states, mapping start states to end states that a program could reach. The program constructs include assignment, for example, $\mathbf{x} := \mathbf{e_1}$ which instantaneously assigns expression $\mathbf{e_1}$ to variable $\boldsymbol{x}$. In the special case of nondeterministic assignment, $\boldsymbol{x} := *$, the transition relation accounts for any possible real value being assigned to $\boldsymbol{x}$, which is useful to describe that arbitrary effects may now be chosen.

The test operator, as in ? **F**, aborts the current run if formula **F** is false, which can be used to impose conditions on the program runs. The continuous evolution operator, $\{\mathbf{x}' = \mathbf{f}(\mathbf{x})\&\mathbf{Q}\}$ follows the ordinary differential equation (ODE) $\mathbf{x}' = \mathbf{f}(\mathbf{x})$ for some nondeterministic amount of time, with evolution domain constraint **Q** being true throughout the evolution. Sequential composition, $\boldsymbol{\alpha}; \boldsymbol{\beta}$, runs program $\boldsymbol{\alpha}$ followed by program $\boldsymbol{\beta}$, for example, the discrete train controller $\boldsymbol{\alpha}$ followed by the train's ODE $\boldsymbol{\beta}$. The nondeterministic choice operator $\boldsymbol{\alpha} \cup \boldsymbol{\beta}$ runs either program $\boldsymbol{\alpha}$ or $\boldsymbol{\beta}$, for example, either accelerate the train with $\boldsymbol{\alpha}$ or brake with $\boldsymbol{\beta}$. When $\boldsymbol{\alpha}$ or $\boldsymbol{\beta}$ contain test conditions ? **F**, then those conditions have the effect of controlling under what circumstances $\boldsymbol{\alpha}$ can be chosen, and when $\boldsymbol{\beta}$ can be chosen, like the if/else branch conditions in most programming languages. The loop operator $\boldsymbol{\alpha}^*$ runs hybrid program $\boldsymbol{\alpha}$ any nondeterministically chosen $\mathbf{n} \geq \mathbf{0}$ times. When running a train control loop indefinitely, it is important to be able to reason about safety for any length of time into the future. To express safety properties about hybrid programs, researchers use the box modality $[\boldsymbol{\alpha}]\mathbf{F}$, which is true in any state from which all runs of hybrid program $\boldsymbol{\alpha}$ end in states in which the formula **F** is true.

For proofs of formulas, researchers use inference rules (Platzer, 2010**;** Platzer, 2018). The rules most relevant to the present work are loop (loop), differential invariant (dI), and differential cut (dC).

$$(\text{loop})\ \frac{\Gamma \vdash J \quad J \vdash [\alpha]J \quad J \vdash P}{\Gamma \vdash [\alpha^*]P}$$

$$(\text{dI})\ \frac{\Gamma \vdash e_1 \leq e_2 \quad Q \vdash [x' := f(x)]e_1' \leq e_2'}{\Gamma \vdash [x' = f(x)\&Q]e_1 \leq e_2}$$

$$(\text{dC})\ \frac{\Gamma \vdash [x' = f(x)\&Q \wedge C]P \quad \Gamma \vdash [x' = f(x)\&Q]C}{\Gamma \vdash [x' = f(x)\&Q]P}$$

We apply induction, which is a general verification technique that allows us to reason about any number of repetitions and any duration of physical motion. An induction rule uses an invariant **J** that holds initially, inductively after each step, and implies the post condition that we seek to prove. The rule establishes that the formula **J** is always true after any number of repetitions of the loop $[\boldsymbol{\alpha}]^*$ and can, thus, be used to justify its safety. A differential invariant preserves properties along the flow of a differential equation: if $\mathbf{e_1} \leq \mathbf{e_2}$ initially and $\mathbf{e_1}$ grows slower than $\mathbf{e_2}$, so $\boldsymbol{e_1'} \leq \boldsymbol{e_2'}$ (where $\boldsymbol{e_1'}$ and $\boldsymbol{e_2'}$ are evaluated after substituting in the assignment $\boldsymbol{x' := f(x)}$ from the differential equation), then it remains true that $\mathbf{e_1} \leq \mathbf{e_2}$. A more general rule form and its explanation can be found in the literature (Mitsch & Platzer, 2011). The idea behind a differential cut is that if formula **C** holds true at the end of every possible run of differential equation $\mathbf{x' = f(x)}$, then **C** must hold true throughout its evolution. Differential cuts can be used to accumulate knowledge about a differential equation.

The semantics of dL is a Kripke semantics in which the states of the Kripke model are the states of the hybrid system. A state is a map ω : V → R, assigning a real value ω(x) to each variable x ∈ V in the set of variables V. We write [Q] to denote the set of states in which formula Q is true, ω ∈ [Q] if formula Q is true at state ω, ω[e] to denote the real value of term e in state ω, and ωe to denote the state ν that agrees with ω except that ν(x) = ω[e]. We write FV(P) for the set of free variables in formula P, and BV(α) to denote the bound variables of program α (Platzer, 2017). The semantics of hybrid programs is expressed as a transition relation [α] [M25]. The differential equations and nondeterministic alternatives in hybrid programs make them an expressive specification language but require computationally expensive methods like online reachability analysis for execution, which is detrimental to their use in reward functions in reinforcement learning. Next, ModelPlex (Mitsch & Platzer, 2014) is reviewed as a method to shift much of this computational complexity offline.

2.2 ModelPlex

ModelPlex [M22] combines a universal offline safety proof [α] P with an existential reachability check whether two concrete states ω, ν are connected by the program α, i.e., whether (ω, ν) ∈ [α]. The safety proof witnesses that all states reachable by model α satisfy P, while passing the reachability check witnesses that the two concrete states ω, ν are connected by the program α, and so state ν inherits the safety proof, i.e., ν ∈ [P]. The reachability check is equivalently phrased in dL as a monitor specification ⟨α⟩ x∈BV(α)(x = x+) (Mitsch & Platzer, 2014). The dL monitor specification allows ModelPlex, in contrast to online reachability analysis, to shift computation offline by using theorem proving to translate a hybrid systems model into a propositional ModelPlex formula over arithmetic expressions. Note that the reachability check is inherently a property of the runtime execution and, therefore, the dL monitor specification and the resulting ModelPlex formula are never valid (they introduce fresh variables x+, which means the existential reachability proof will not succeed offline for all states). Instead, the proof can be finished at runtime for two concrete states (a state ω providing values for x and a state ν providing values for x+) by plugging in concrete measurements for all variables of the ModelPlex formula.

The set M of ModelPlex formulas ϕ : S×S → B is generated by the following grammar (∼ ∈ {≤, <, =,≠, >, ≥} and θ, η form the arithmetic expressions of the set T of ModelPlex terms in +, −, ·, / over the reals, i.e., θ : S × S → R):

$$\phi, \psi ::= \theta \sim \eta \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi$$

When a ModelPlex formula $\phi \in M$ is satisfied over states ω, ν, we write $(\omega, \nu) \models \phi$ as shorthand for $\omega\nu(x) \in [\phi]$, or in other words, $\phi(\omega, \nu)$ is true. ModelPlex formulas are quantifier- and program-free and are therefore easy (and computationally inexpensive) to evaluate from concrete measurements at runtime, which makes them attractive for use in reinforcement learning.

The predictive nature of ModelPlex monitors also makes them useful for safeguarding learned controllers during training and during operation (Fulton & Platzer, 2018 & 2019), in a shielding-like approach (Konighofer et al., 2020) based on hybrid systems models. In this research, the team took a complementary approach to shielding by interpreting ModelPlex monitors quantitatively in rewards.

2.3 Reinforcement Learning and Reward Shaping

Reinforcement learning involves training an agent to reach a goal by allowing the agent to explore an environment while trying to maximize its reward. The agent attempts different actions from the set of actions (a). For every action the agent makes, the environment makes a transition from state $s \in S$ to a new state $s' \in S$. A reward function then signals to the agent the usefulness of the outcome of action a: a negative reward signals that taking action a to reach state s' is discouraged, while a positive reward encourages action a. Put differently, negative rewards encourage leaving "bad" states, while positive rewards encourage dwelling in "good" states. Section 4 details how the team developed a principled approach to generate safety-oriented aspects of the reward function from hybrid systems models. Often, the agent must learn multiple (i.e., competing) aspects of control (e.g., reaching a destination fast while respecting posted speed limits and conserving energy). This process of augmenting the reward function with multiple aspects is referred to as reward shaping. The motivation behind reward shaping is to provide additional reward for accomplishing subtasks that could lead toward the goal to improve convergence and efficiency of training.

2.4 Model of Train Kinematics

This section introduces the kinematics model, which provides the forces acting on a train (Brosseau & Ede, 2009). After the net force on the train has been identified, Newton's second law, using train mass, determines the acceleration that the train experiences, which in turn determines change in velocity and change in position for train control design. The forces are

$$\sum \boldsymbol{F} = -\boldsymbol{F_G} - \boldsymbol{F_C} - \boldsymbol{F_B} + \boldsymbol{F_L} - \boldsymbol{F_R} = \boldsymbol{m_T a} \quad \textbf{(1)}$$

where F_G denotes force due to track grade, F_C resistance due to track curvature, F_B force from the brakes, F_L force from the locomotive engine (i.e., tractive effort), F_R resistive forces, and m_T, train mass. Newton's second law, $\sum F = m_T a$, determines train acceleration a. Resistive force F_R follows the modified Davis equation (Brosseau & Ede, 2009):

$$\boldsymbol{F_R = A_R w + B_R n + C_R w v + D_R v^2},$$

where A_R, B_R, C_R, D_R are experimentally determined positive constants, whose numerical value given a choice of units can be found in other sources (Brosseau, et al., 2013; Hay, 1982; Dasher

& Morrison, 2020). Further, v is the velocity of the train, n is the number of axles, and w is the weight of the train.

Grade and curve forces depend on the train position $\mathbf{p}$ on the track. Grade force (Brosseau & Ede, 2009) is proportional to train weight $\mathbf{w}$ and average track grade $\boldsymbol{grade(p)}$ underneath the train:

$$\boldsymbol{F_G = A_G w \cdot grade(p)}$$

Similarly, curve force (Brosseau & Ede, 2009) is a function of average track curvature $curve(p)$ along the train and train weight w:

$$\boldsymbol{F_C = A_C w \cdot curve(p)}$$

where A_C and A_G are positive multiplicative constants. Braking force can be modeled as the minimum of two linear functions to capture the effect of air pressure brake force buildup (Brosseau & Ede, 2009) and stabilization. Let F_{B0} be the force acting immediately on brake application, f_p the slope with which brake force increases, t the elapsed time, and F_{Bmax} the force after air pressure in the air pressure brakes saturates.

$$\boldsymbol{F_B = \min\left(F_{B0} + f_p t, F_{Bmax}\right)}$$

Brake enforcement and train protection algorithms (Brosseau, et al., 2013; Brosseau & Ede, 2009) approximately solve a differential equation derived from Eq. (1) to estimate the velocity and position of the train at future times.

2.5 Mathematical Model Abstraction

To maximize generality of the embedded software design, minimize verification effort, and simplify future proof maintenance, the team used a mathematical abstraction of the kinematics model (Brosseau & Ede, 2009; Brosseau, et al., 2013). Concrete verified controllers and their safety proofs for the fully expanded model can be obtained automatically from the verified abstract model by uniform substitution (Platzer, 2017).

The abstract train kinematic model in Eq. (2) is an ODE in time. The rate of change of position is velocity, and the rate of change of velocity is acceleration. The variables and constants involved, along with their signs, when relevant, are (i) train position $\boldsymbol{p}$, (ii) train velocity $\boldsymbol{v}$, (iii) velocity and position-independent component of acceleration $\boldsymbol{a_l}$ ranging from immediate braking ability $-\boldsymbol{b}_{\max} < \mathbf{0}$ to maximum train engine acceleration $\boldsymbol{a}_{\max} > \mathbf{0}$, (iv) acceleration due to air brakes $\boldsymbol{a_a}$ in range $\boldsymbol{a_{b\max}} < \mathbf{0}$ to 0, (v) rate of change $\boldsymbol{m_b}$ of air brake acceleration, which is $\boldsymbol{m_p} < \mathbf{0}$ when brakes are ramping up and 0 otherwise, (vi) map $\boldsymbol{a_s}$ from position to acceleration due to grade, (vii) map $\boldsymbol{a_c}$ from position to acceleration due to curvature, and (viii) velocity-dependent resistance $\boldsymbol{a_r}$. In the chosen sign convention, resistive acceleration is negative.

(2)

$$\boldsymbol{p' = v, v' = a_l + a_a + a_s(p) + a_r(v) + a_c(p), a_b' = m_b}$$

$$\text{with } \boldsymbol{a_l} \in [-\boldsymbol{b_{max}}, \boldsymbol{a_{max}}], \boldsymbol{a_a} = \max(\boldsymbol{a_b}, \boldsymbol{a_{bmax}}), \boldsymbol{m_b} \in \{\mathbf{0}, \boldsymbol{m_p}\}$$

The Davis equation resistance $a_r(v) = -\frac{C_R w v + D_R v^2}{m_T}$ has the shape $a_r = a_1 v + a_2 v^2$ when $a_1 = -C_r g$ with gravity g summarizes the linear coefficient of velocity, and $a_2 = -\frac{D_R}{m_T}$ summarizes the quadratic coefficient. Grade and curvature are represented by unspecified but bounded functions a_s and a_c that map train positions to a numeric value for acceleration due to slope and average curvature, respectively. The quantity a_l summarizes locomotive tractive effort ($a_l \geq 0$) and train deceleration ($a_l < 0$) as commanded by the train controller, with adjustment for the velocity-independent resistance.

The team later instantiated the proved abstract kinematic train model by dL's uniform substitution (Platzer, 2017) to easily get proofs for specific physical train models. Similarly, proofs for specific train configurations result also from substituting values for coefficients, or even for a specific train state, when additionally substituting speed and position.

3. Model Structure

The team developed a conservative train controller based on the abstract train kinematic model Eq. (2) of Section 2.2, presenting it modularly and introducing conceptually important model components and functions along the way. Researchers addressed the challenge of representing track grade and curve, which are unknown at proof time, using unspecified maps. To reason about them, researchers bound the maps with assumptions quantifying over all arguments. This solution permit the team to capture the full complexity without conservatively neglecting $a_s(p)$ and $a_c(p)$ when reasoning about it during verification. It generalizes to other embedded software that must reason about unspecified, bounded functions, such as noise or potential fields (e.g. electro-magnetic or gravitational effect).

The controller safe was proven safe: relative to Eq. (2), the controller provably will never permit the train's position to exceed the EOA $\boldsymbol{e}$, though it might be inefficient, braking unnecessarily early. Later, by revising modular components and functions to be more arithmetically sophisticated, Section 4 will show how provable safety is retained but the controller is made more efficient. Figure 1 shows the relationship between the resulting controller models.

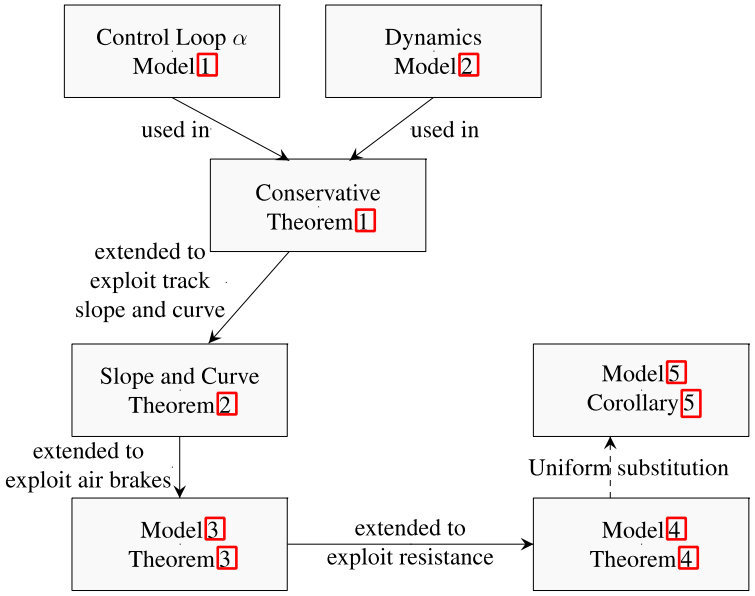


Figure 1. Relationship between train models in this report.

The train controller consists of a time control loop. The control has a latency of time $T > 0$ – the controller must wait this long at most before being able to change the brake position. In practice, reaction time T is typically to the order of 1 second, but train controllers often keep decisions in effect for a 10 second period without revising them (Dasher & Morrison, 2020). Every control cycle, the controller computes an over approximation $\text{stopDist}(p, v, a_b)$ of the *stopping distance*, the distance that the train will travel before stopping if it were to accelerate during the current control cycle, but then brake continuously starting at the next control cycle, until it comes to a halt. If the distance left to the EOA exceeds $\text{stopDist}(p, v, a_b)$, the controller continues free driving (with any acceleration or deceleration choice within the physical limits of the train);

otherwise, it brakes. This control cycle, $\alpha(\text{stopDist})$, is parametric in stopping distance stopDist, and expressed as a hybrid program in Model 1.

Model 1 Train control $\alpha(\texttt{stopDist})$.

Reset Timer	1	$t := 0;$
	2	$\big($ $(?(e - p > \texttt{stopDist}(p, v, a_b));$
Free Driving	3	$a_l := *;\ ? - b_{\max} \leq a_l < a_{\max};$
	4	$a_b := 0;\ m_b := 0)$
Braking	5	$\cup\ a_l := -b_{\max};\ m_b := m_p\big)$

3.1 Train Dynamics

Model 2 describes the physical motion of the train according to the abstract mathematical model of Eq. (2) augmented with a clock $t' = 1$ to switch back to control after at most time T. *All constants are left symbolic and safety is proved for all values*, so that individual railroads have the flexibility to instantiate them with the values that apply to their system and inherit the safety results. This generality in controller design comes at the cost of higher proof complexity, but compared to the alternative where these constants would be initialized with a conservative value, this allows for more efficient, tailored controllers adapted to the specific rail operation.

Model 2 Train dynamics.

Dyn.	1	$p' = v, a_b' = m_b, t' = 1,$
	2	$v' = a_l + \max(a_b, a_{b\max}) + a_s(p) + a_r(v) + a_c(p)$
	3	$\&\ t \leq T \wedge v \geq 0$

3.2 Stopping Distance

To decide between free driving and braking, the controller computes the upper bound $\text{stopDist}(\boldsymbol{p}, \boldsymbol{v}, \boldsymbol{a}_\text{b})$ on the distance covered over one period of acceleration and subsequently braking to a stop.

Thus, the models provide two distinct specifications of the distance that the train will take to stop. The first, indirect specification is through the differential equation that implicitly describes the physical motion of the train. The second, approximate specification is stopDist in line 2 of Model 1, an explicit arithmetic expression that the controller can evaluate efficiently to make decisions at runtime.

Efficiency concerns demand that $\text{stopDist}(\boldsymbol{p}, \boldsymbol{v}, \boldsymbol{a}_\text{b})$ be as tight as possible; if the bound is too large, the controller would enforce braking unnecessarily. But verifiable safety requires $\text{stopDist}(\boldsymbol{p}, \boldsymbol{v}, \boldsymbol{a}_\text{b})$ provably to be an upper bound on the distance that the train covers (as determined by the dynamics). The tightest possible bound is the exact solution of the differential equation. However, even ignoring the effect of air brakes, the differential equation requires

trigonometric solutions. Transcendental function arithmetic is undecidable (Richardson, 1968). To ensure mathematical provability[1], polynomial approximations are developed, which is a delicate design task because automated decision procedures for polynomial real arithmetic validity are computationally expensive (Davenport & Heintz, 1988**;** Weispfenning, 1988). This constrains the complexity of the polynomial approximations that can be used as upper bounds. A balance is therefore struck between conflicting concerns: striving for efficiency while satisfying mathematical provability.

To illustrate this approach, start with a simple conservative expression for stopDist. This expression is similar in complexity to previous work (Platzer & Quesel, 2009**;** Mitsch, et al., 2017), but is now proved safe for the full model including slope, curve friction, air brake propagation, and aerodynamic drag. This approximation will be improved upon later, with the focus on one contributing factor at a time.

3.3 Stopping Distance: Conservative

In this section, a first, conservative controller is constructed by instantiating control loop α with an expression for stopDist and proving it safe. Referring to the train dynamics in Eq. (2), an upper bound for v is first needed. Integrating this bound via $p' = v$ computes a stopping distance upper bound.

The first impediment to obtaining a provable upper bound for $\boldsymbol{v}$ is that track profiles with grade and curvature information $\boldsymbol{a_s}$ and $\boldsymbol{a_c}$ are arbitrary functions, constrained only by upper and lower bounds, and bounded gradients. At runtime, the train knows their exact values since the controller is instantiated with profiles for the railroad upon which it runs. These profiles are unknown at proof time, but the proof must show safety of the train control ahead of time for *all* possible track profiles to justify safety of the train controller. To obtain a provable upper bound on stopping distance, the proof therefore is based on the limited information available about the maps: upper bounds on the potential values of $\boldsymbol{a_s}$ and $\boldsymbol{a_c}$.

A naïve upper bound on $\boldsymbol{a_s}$ is the value of acceleration that the train experiences when it is on the steepest permissible downward slope, $\boldsymbol{m_s}$. The proof will show that the distance required to stop for any permissible grade map cannot exceed the distance computed with the steepest downward slope. It first shows that the true acceleration is bounded above by an acceleration that uses the highest permissible value of grade acceleration, then that actual velocity cannot exceed the velocity computed using the worst-case acceleration, and consequently, that traveled distance cannot exceed the stopping distance computed using the worst-case estimate of velocity.

Accounting for grade force is important. On a downward hill, for example, a train with a controller that ignores grade would roll forward even at the EOA which could cause accidents. In contrast, the curve resistance can be ignored safely when approximating stopDist, since resistances shorten stopping distance (upper bound 0). These simplifications result in differential equation $\boldsymbol{v_2' = a_l + m_s + a_r(v)}$ (where $\boldsymbol{a_r(v)}$ is $\boldsymbol{a_1 v + a_2 v^2}$), since $\boldsymbol{max(a_b, a_{b\max}) = 0}$ while the train is accelerating, where $\boldsymbol{v_2}$ is the upper bound on $\boldsymbol{v}$ that is integrated to compute stopDist. However, the solution $\boldsymbol{v_2(t)}$ is still transcendental:

[1] In KeYmaera X, even when manually simplifying differential and modal expressions, arithmetic subgoals are outsourced to arithmetic decision procedures, which are subject to limitations in proving real arithmetic.

$$\left(\sqrt{4(a_l + m_s)a_2 - {a_1}^2}\right) \cdot \frac{\tan\left(t\frac{\sqrt{4(a_l + m_s)a_2 - {a_1}^2}}{2} + \tan^{-1}\left(\frac{a_1 + 2a_2v_0}{\sqrt{4(a_l + m_s)a_2 - {a_1}^2}}\right)\right) - a_1}{2a_2}$$

The culprits are the linear and quadratic terms in velocity from the Davis equation. With another simplification of 0 as an upper bound for a_r(resistance always works against the train's motion), a polynomial expression is derived for stopDist:

(3)

$$\mathbf{v_3 = a_l + m_s} \implies \mathbf{v_{3(t)} = v_0 + (a_l + m_s) \cdot t}$$

$$\boldsymbol{p_3' = v} \implies \boldsymbol{p_{3(t)} = p_0 + \left(v_0 + \frac{a_l + m_s}{2} \cdot t\right) \cdot t}$$

where v_0 and p_0 are the initial values of speed and position. The solutions provide a conservative stopping distance bound.

(4)

$$\text{stopDist}_b(p, v, a_b) = \underbrace{vT + \frac{a_{max} + m_s}{2} \cdot T^2}_{\text{distance while accelerating}} + \underbrace{\frac{(v + (a_{max} + m_s)T)^2}{2(b_{max} - m_s)}}_{\text{stopping from increased speed}}$$

The conservative stopping distance stopDist_b ignores its arguments p and a_b, but later refinements of stopDist functions also depend on p and a_b, which is why they are passed in.

The first two terms are the distance covered by the train in one control cycle of acceleration, while the third term is the distance that the train needs to stop should it start braking right after, assuming the worst-case value of 0 for $\boldsymbol{a_a}$. This conservative distance is like what has been used in the literature (Platzer & Quesel, 2009), but has been adjusted to account for grade force with its worst-case accelerating or decelerating effects. Substituting into control cycle $\boldsymbol{\alpha}$ results in the hybrid program of the conservative controller: $\boldsymbol{\alpha}(\text{stopDist}_{\boldsymbol{b}})$.

A physically important quantity is *braking distance*, the distance that the train will travel before coming to a stop should it start braking immediately. An upper bound is derived, $\text{brakeDist}_{\boldsymbol{b}}(\boldsymbol{v}, \boldsymbol{a}_\text{b})$, which will be crucial to the proofs and the initial assumptions, and is improved upon later.

(5)

$$\text{brakeDist}_{\boldsymbol{b}}(\boldsymbol{v}, \boldsymbol{a}_\text{b}) = \frac{\boldsymbol{v^2}}{\boldsymbol{2(b_{\max} - m_s)}}$$

The last term of Eq. (4) is $\text{brakeDist}_b(v_3(T), 0)$ with v_3 according to Eq. (3) when $a_l = a_{\max}$.

Initial assumptions init(brakeDist) parametrized by brakeDist, and initAirbrake (assumptions on air brakes) are required to prove the conservative controller safe. Assumptions about unspecified functions are represented by universal quantification over their input. This representation permits derivation of a formula about the unspecified function at any point of the train's evolution by substituting the quantified input with current values.

(6)

$$\text{init(brakeDist)} = \boldsymbol{a_{max} > 0 \wedge b_{max} > 0 \wedge a_1 < 0 \wedge a_2 < 0}$$

$$\wedge \boldsymbol{e - p} > \text{brakeDist}(\boldsymbol{v}, \boldsymbol{0}) \wedge \boldsymbol{b_{max} - m_s > 0 \wedge v \geq 0}$$

$$\wedge \boldsymbol{m_s \geq 0 \wedge T > 0 \wedge \forall x(|a_s(x)| \leq m_s) \wedge \forall x(a_c(x) \leq 0)}$$

$$\text{initAirbrake} = \boldsymbol{m_p < 0 \wedge a_{bmax} < 0 \wedge m_b = 0 \wedge a_b = 0}$$

Theorem 1 presents the formula representing the safety of the conservative controller.

Theorem 1. *The conservative braking controller guarantees that the train always remains within the EOA. The* dL *formula below is provable, where* $\boldsymbol{\alpha}$ *is the control loop from Model 1 parameterized with Eq. (4) for stopDist.*

$$\text{init}(\text{brakeDist}_{\boldsymbol{b}}) \wedge \text{initAirbrake} \rightarrow [(\boldsymbol{\alpha}(\text{stopDist}_{\boldsymbol{b}}); \boldsymbol{Model\ 1})*]\boldsymbol{e - p > 0}$$

Proof. The proof has been done in the theorem prover KeYmaera X, but its central ideas are presented here. Loop invariant $\boldsymbol{e - p} \geq \text{brakeDist}_{\boldsymbol{b}}(\boldsymbol{v}, \boldsymbol{0}) \wedge \boldsymbol{a}_{\text{b}} \leq \boldsymbol{0} \wedge \boldsymbol{v \geq 0}$ is used and split into cases for free driving and braking corresponding to the nondeterministic choice in Lines 2–5 of Model 1. On braking, the invariant is maintained because the derivative of the distance that the train will take to come to a stop does not exceed the derivative of the distance to the EOA, i.e., $\text{stopDist}_{\boldsymbol{b}}' \leq (\boldsymbol{e - p})'$, by the dI rule. On free driving for a control period $\boldsymbol{T}$, first it is restated that the train maintains a distance to the EOA of at least stopDist adjusted for time $\boldsymbol{t}$ since the last control decision, i.e., $\boldsymbol{v(T - t)} + \frac{\boldsymbol{a}_{\max}+\boldsymbol{m_s}}{\boldsymbol{2}}\boldsymbol{(T - t)^2} + \frac{\boldsymbol{(v+(a}_{\max}+\boldsymbol{m_s)(T-t))^2}}{\boldsymbol{2(b}_{\max}-\boldsymbol{m_s)}}$ by dC and dI. The required inequality relation between the derivatives, $\boldsymbol{-v \geq -v + v'(T - t) - (a_{max} + m_s)(T - t) + \frac{(v+(a_{max}+m_s)(T-t))(v'-(a_{max}+m_s))}{b_{max}-m_s}}$, holds because $\boldsymbol{v' - (a_{max} + m_s) \leq 0}$.

4. Safe Efficiency Improvements

The team improved on the over approximation for stopping distance to make the controller more efficient. The FRA model presents two challenges common in embedded controllers: it uses functions whose exact values are unknown at proof time (track profiles with slope and curve information) and has many interacting forces. The team's techniques address these problems with two general principles: using quantified worst case bounds on unknown functions, and separation of dependencies. The first technique relies on the observation that the track changes are gradual and predictable (i.e., the rate of change of unknown functions is bounded). It drastically improves bounds on the effect of grade and curve over one period of acceleration, after resolving *circular dependencies* between the variables of motion. The second technique improves the estimate brakeDist by accounting for air brake dynamics. It demonstrates a handling of triple integration, using *mode-splitting* to deal with the non-analytical change of behavior when brakes saturate. The third technique uses *Taylor polynomials* to capture the effect of resistance, which would otherwise lead to transcendental arithmetic. The track environment discussion in Section 4.1 and 4.2 will become assumptions of these models, while the calculations in Section 4.3 and 4.4 are machine-checked to be correct approximations as part of the KeYmaera X proofs.

4.1 Bound on Gradient

The train controller knows the current slope $a_s(p)$ and vertical curves of the track, which determine transitions from one track grade to another. This knowledge results in a bound $h_{\max}$ on the difference in grade per unit length (Hay, 1982):

$$\left|\frac{\partial a_s(x)}{\partial x}\right| \leq h_{max} \quad \Rightarrow |a_s'(p)| \leq v h_{max}$$

The second inequality follows from the first using the chain rule and $p' = v$. After time T, a_s could have increased by no more than $u h_{\max} T$, where u is some upper bound on v over the course of T time, which is derived later in this section.

4.2 Bound on Curve Resistance

As with bounding the gradient change, an upper bound is computed on the rate of change of curve resistance as a function of velocity using track geometry. Curve resistance depends on average curve *curve(p)*.[2] Assume that the tightest permissible curvature for the railroad corresponds to radius r. The greatest change in average curvature occurs when a train goes from a track with the greatest permissible curvature to a straight track (or vice versa). Over a small period dt, the portion of the train transitioning from greatest curvature to 0 curvature is dv, where v is velocity. So, the rate of change of curve(p) (taken in radians) with respect to time is

[2] In practice, degree of curvature is the angle subtended by a 100ft arc of the track, 100ft chord of the track, or 100.7ft arc in the US. In Europe, radius is generally used (Hay, 1982). In this rate of change of curvature derivation, the mathematical idealization for average degree of curvature is used: $\text{curve}(p) = \int_{p-l}^{p} \frac{1}{lr(x)} dx$ where l is the length of the train and $r(x)$ is the radius of curvature at track point x. This approximately relates to the US definition by a multiplicative factor, with a small error introduced by the granularity of the 100ft measuring arc. This multiplicative factor can be rolled into the multiplicative coefficient c_{cft}.

$-\frac{\mathrm{v}}{\mathrm{lr}}$ where l is the length of the train. For a given train, $\mathrm{a_c}$ relates to curve with some constant multiplicative factor q. $a_c' = q\frac{v}{lr} = \mathrm{c_{cft}}v$ is used with the constant factor $\mathrm{c_{cft}} = \frac{\mathrm{q}}{\mathrm{lr}}$.

With this bound on the maximum rate of change of $\mathbf{a_c}$, the upper bound on curve resistance over time $\mathbf{T}$, where $\mathbf{p_0}$ is initial train position, is now estimated to be

$$\mathbf{a_c}(\mathbf{p_0}) + \mathbf{u}\mathbf{c}_{\mathrm{cft}}\mathbf{T}$$

As before, u is an upper bound on velocity for duration T.

4.3 Tight Stopping Distance Approximation

The upper bounds $\overline{a}_s$ on gradient from and $\overline{a}_c$ on curve resistance from are summarized as:

$$\begin{aligned} \boldsymbol{a_s}(\boldsymbol{p}) &\leq \overline{\boldsymbol{a}}_{\boldsymbol{s}}(\boldsymbol{p_0}) = \mathbf{min}(\boldsymbol{m_s}, \boldsymbol{a_s}(\boldsymbol{p_0}) + \boldsymbol{u}\boldsymbol{h}_{\max}\boldsymbol{T}) \\ \boldsymbol{a_c}(\boldsymbol{p}) &\leq \overline{\boldsymbol{a}}_{\boldsymbol{c}}(\boldsymbol{p_0}) = \mathbf{min}(\mathbf{0}, \boldsymbol{a_c}(\boldsymbol{p_0}) + \boldsymbol{u}\boldsymbol{c}_{\mathrm{cft}}\boldsymbol{T}) \end{aligned}$$

This improves the estimation of stopping distance:

$$\boldsymbol{v_4'} = \boldsymbol{a_l} + \overline{\boldsymbol{a}}_{\boldsymbol{s}}(\boldsymbol{p_0}) + \overline{\boldsymbol{a}}_{\boldsymbol{c}}(\boldsymbol{p_0}) \Rightarrow \boldsymbol{v_4}(\boldsymbol{t}) = \boldsymbol{v_0} + \left(\boldsymbol{a_l} + \overline{\boldsymbol{a_s}}(\boldsymbol{p_0}) + \overline{\boldsymbol{a}}_{\boldsymbol{c}}(\boldsymbol{p_0})\right)\boldsymbol{T} \quad \mathbf{(7)}$$

Upper bound v_4 is tighter than v_3 of Eq. (3) and thus integrates to an improved stopping distance estimate. It depends (transitively through $\overline{a}_s$ and $\overline{a}_c$) on the unknown upper bound u on velocity, which is needed to estimate provability.

4.3.1 Circular Dependencies

The upper bound on velocity, $\boldsymbol{u}$, is undefined in expression (7) above. We cannot use the bound $\boldsymbol{v_4}$ for $\boldsymbol{u}$, since $\boldsymbol{v_4}$ itself is phrased in terms of $\boldsymbol{u}$. The problem is a circular dependency between $\boldsymbol{a_s}$ and $\boldsymbol{v}$: the bound on slope acceleration $\boldsymbol{a_s}$ depends on speed $\boldsymbol{v}$, while the upper bound on speed $\boldsymbol{v}$, in turn, depends on slope acceleration $\boldsymbol{a_s}$; likewise, for $\boldsymbol{a_c}$. Physically, this is because if the train is moving faster, less is known about the nature of the track – its curve and slope – after the passage of some time, as the train is farther from its previous position on the track. However, information is needed about the grade curve to better estimate the velocity that the train is traveling at. To cut through these circular dependencies, the conservative estimations of these quantities from Eq. (3) are used as a base case to bootstrap incrementally finer computations, as presented below.

We first use the initial upper bounds $\boldsymbol{m_s}$ for $\boldsymbol{a_s}$ and 0 for $\boldsymbol{a_c}$ to get a conservative bound $\boldsymbol{v}(\boldsymbol{t}) \geq \boldsymbol{v_0} + (\boldsymbol{a}_{\max} + \boldsymbol{m_s})\boldsymbol{t}$, so that we can set $\boldsymbol{u} = \boldsymbol{v_0} + (\boldsymbol{a}_{\max} + \boldsymbol{m_s})\boldsymbol{T}$. Since $(\boldsymbol{a}_{\max} + \boldsymbol{m_s})$ is a positive upper bound on the train's acceleration, velocity could have increased no more than $(\boldsymbol{a}_{\max} + \boldsymbol{m_s})\boldsymbol{T}$. Hence $\boldsymbol{u}$ is indeed an upper bound on $\boldsymbol{v}$ through the $\boldsymbol{T}$ time interval. Substituting this $\boldsymbol{u}$ refines the gradient and curve resistance bounds.

$$\begin{aligned} \overline{\boldsymbol{a}}_{\boldsymbol{s}}(\boldsymbol{p_0}) &= \mathbf{min}\left(\boldsymbol{m_s}, \boldsymbol{a_s}(\boldsymbol{p_0}) + \overbrace{(\boldsymbol{v_0} + (\boldsymbol{a}_{\max} + \boldsymbol{m_s})\boldsymbol{T})}^{\boldsymbol{u}}\boldsymbol{h}_{\max}\boldsymbol{T}\right) \\ \overline{\boldsymbol{a}}_{\boldsymbol{c}}(\boldsymbol{p_0}) &= \mathbf{min}\left(\mathbf{0}, \boldsymbol{a_c}(\boldsymbol{p_0}) + \underbrace{(\boldsymbol{v_0} + (\boldsymbol{a}_{\max} + \boldsymbol{m_s})\boldsymbol{T})}_{\boldsymbol{u}}\boldsymbol{c}_{\mathrm{cft}}\boldsymbol{T}\right) \end{aligned}$$

These expressions give the chosen definitions of $\overline{a}_s$ and $\overline{a}_c$ by replacing placeholder velocity bound u.

We could in principle further improve this upper bound on speed by using $\boldsymbol{v_4}$ to obtain an even better bound on $\boldsymbol{a_s}$ and $\boldsymbol{a_c}$, which could in turn yield an improved bound on $\boldsymbol{v}$. One could perform an unbounded number of extra iterations to get more efficient control. However, extra levels of extrapolation increase proof cost and computation time when the controller is run. Each extra intermediate bound requires a constant number of extra proof steps, but provides diminishing efficiency gains in return. Intuitively, proof length is asymptotically linear in number of iterations because under optimal proof rule application ordering, each iteration induces one extra application of rule to introduce the intermediate bound into the proof tree, and rule to justify this intermediate bound. For railroads with steep slopes and sharp curves, the benefit of performing extra iterations of dependency propagation would increase. The benefits of obtaining a controller with a more complex, tighter bound on stopping distance might exceed the cost of extra proof steps.

The stopDist expression below uses $\boldsymbol{v_4}$ with $\boldsymbol{u = v_0 + (a_{\max} + m_s)T}$ to estimate stopping distance, which is sufficiently tight to make useful control decisions (see Section 6).

$$\textbf{stopDist}_{\boldsymbol{s}}(\boldsymbol{p, v, a_b}) = \boldsymbol{vT} + \left(\frac{\boldsymbol{a_{max}+\overline{a}_s(p)+\overline{a}_c(p)}}{\boldsymbol{2}}\right)\boldsymbol{T^2} - \frac{\boldsymbol{\left(v+\left(a_{max}+\overline{a}_s(p)+\overline{a}_c(p)\right)T\right)^2}}{\boldsymbol{2(b_{max}-m_s)}} \qquad \textbf{(8)}$$

We need further initial assumptions to prove the improved *slope-exploiting controller* $\alpha(\text{stopDist}_s)$ safe. These assumptions represent the result of the track environment discussion of Section 4.1 and Section 4.2 used for the computer-checked proof.

$$\begin{aligned} &\text{init}_s(\text{brakeDist}) \equiv \text{init}(\text{brakeDist}) \\ &\wedge \boldsymbol{a_{max} - m_s + m_c > 0 \wedge h_{max} \geq 0} \\ &\wedge \boldsymbol{m_c \leq 0 \wedge c_{cft} \geq 0 \wedge \forall x(a_c(x) \geq m_c)} \\ &\wedge \boldsymbol{\forall x' \forall x\left(|a_c'| \leq x' c_{cft}\right) \wedge \forall x' \forall x(|a_s'| \leq x' h_{max})} \end{aligned} \qquad \textbf{(9)}$$

The technique in this section applies to *time-triggered controllers* (where a control loop runs with some known maximum latency and sensors measure current state every cycle) for physical systems with functions affecting the environment that are unknown except for bounds on their rate of change. The future value of the functions can be bounded in terms of their worst-case rate of change. Furthermore, these bounds can be used to compute bounds on other variables in the system, just as here a bound on velocity was used to bound slope and curve effect, which was again used to obtain a better bound on velocity. The situation arises frequently in practice: examples of unknown functions are a potential field, or a noise or error effect, which may have circular dependence with position.

4.3.2 Safety of the Slope-Exploiting Controller

Theorem 2. *The slope-estimating controller guarantees that the train stays within its MA. The formula below is provable, where* $\boldsymbol{\alpha}$ *is the control loop from Model 2 parameterized with Eq. (8) for stopDist.*

$$\text{init}_{\boldsymbol{s}}(\text{brakeDist}_{\boldsymbol{b}}) \wedge \text{initAirbrake} \rightarrow \left[\left(\boldsymbol{\alpha}(\text{stopDist}_{\boldsymbol{s}}); \boldsymbol{Model\ 2}\right)*\right]\boldsymbol{e - p > 0}$$

Proof. By proof in KeYmaera X. The proof builds on the ideas from Theorem 1. The loop rule with the same loop invariant as Theorem 1 is applied. If the train brakes, differential invariant rule again shows that the loop invariant holds throughout differential equation evolution.

If the train chooses to accelerate, then as before, it is restated that the train maintains at least a distance of $\text{stopDist}_{\boldsymbol{s}}$ adjusted for time $\boldsymbol{t}$ since the last control decision. Unlike before, $\overline{\boldsymbol{a}}_{\boldsymbol{s}}$ instead of $\boldsymbol{m_s}$ accounts for worst-case $\boldsymbol{a_s}$, and $\overline{\boldsymbol{a}}_{\boldsymbol{c}}$ instead of $\boldsymbol{0}$ accounts for worst-case $\boldsymbol{a_c}$. Again, this proves that this adjusted inequality remains true. To prove the required inequality on the derivatives, we first use differential cut rule to show $\overline{\boldsymbol{a}}_{\boldsymbol{s}}(\boldsymbol{p_0}) \geq \boldsymbol{a_s}(\boldsymbol{p})$ throughout the control cycle, and $\overline{\boldsymbol{a}}_{\boldsymbol{c}}(\boldsymbol{p_0}) \geq \boldsymbol{a_c}(\boldsymbol{p})$. There are two branches for each cut corresponding to how the $\mathbf{min}$ in $\overline{\boldsymbol{a}}_{\boldsymbol{s}}$ and $\overline{\boldsymbol{a}}_{\boldsymbol{c}}$ resolve. For example, using initial position and velocity $\boldsymbol{p_0}$ and $\boldsymbol{v_0}$, for $\overline{\boldsymbol{a}}_{\boldsymbol{s}}$, we need to show that $\boldsymbol{a_s}(\boldsymbol{p}) \leq \boldsymbol{m_s}$, and $\boldsymbol{a_s}(\boldsymbol{p}) \leq \boldsymbol{a_s}(\boldsymbol{p_0}) + \boldsymbol{u}\boldsymbol{h}_{\max}\boldsymbol{T}$. While the former follows from the quantified assumption on $\boldsymbol{a_s}(\boldsymbol{p})$, to prove the latter, we again adjust it for elapsed time $\boldsymbol{t}$, to argue that $\boldsymbol{a_s}(\boldsymbol{p}) \leq \boldsymbol{a_s}(\boldsymbol{p_0}) + \big(\boldsymbol{v_0} + \boldsymbol{u}\boldsymbol{h}_{\max}(\boldsymbol{T} - \boldsymbol{t})\big)$, proved using dI. The required derivative inequality follows from instantiating the quantified assumption bounding the rate of change of $\big(\boldsymbol{a_s}(\boldsymbol{p})\big)' \leq \boldsymbol{u}\boldsymbol{h}_{\max}$ with the current position and showing that $\boldsymbol{u}$ is an upper bound on $\boldsymbol{v}$ in the control loop. The argument for $\overline{\boldsymbol{a}}_{\boldsymbol{c}}$ is analogous.

4.4 Effect of Air Pressure Brakes

The term brakeDist_b conservatively neglects the *significant* effects of air brakes to avoid reasoning about their time dependence. In this section, a tighter brakeDist_a is derived that accounts completely for air brakes. It specifies a controller that simultaneously benefits from the slope and curve estimation of the previous section, and from air brake dynamics. The central insight required to prove the improved controller safe is how to compose reasoning about time-dependent air brake propagation and velocity-dependent slope and curve estimations from the previous section. We first show that brakeDist_a, the component of stopDist affected by air brakes, is the desired upper bound on distance to brake throughout the control loop. Then, holding brakeDist_a constant, we perform the differential reasoning on slope and curve estimation described in the previous section. The two results together permit an overall proof of safety of the air brake-exploiting controller. The technique is particularly useful for long freight trains, where air brake propagation has a significant effect.

To derive the improved $\text{brakeDist}_{\boldsymbol{a}}$, we first compute some intermediate functions from air brake dynamics. During brake ramp-up, with slope relaxed pessimistically to $\boldsymbol{m_s}$, and curve and resistance to 0, $\mathbf{max}(\boldsymbol{a}_{\text{b}}, \boldsymbol{a}_{\boldsymbol{b}\max})$ evaluates to $\boldsymbol{a}_{\text{b}}$, and $\boldsymbol{m_b}$ to $\boldsymbol{m_p}$. The solution for $\boldsymbol{v}$ in the resulting differential equation $\boldsymbol{v}' = \boldsymbol{b_{max}} - \boldsymbol{m_s} + \boldsymbol{a_b}, \boldsymbol{a_b}' = \boldsymbol{m_p}$ is quadratic in $\boldsymbol{t}$:

$$\boldsymbol{v} = \boldsymbol{v_0} - (\boldsymbol{b}_{\max} - \boldsymbol{m_s} + \boldsymbol{a}_{\text{b}})\boldsymbol{t} + \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{m_p}\boldsymbol{t}^{\mathbf{2}} \tag{10}$$

Function t_b below computes the time the train takes to achieve full braking by subtracting current brake buildup a_{b} from maximal air braking $a_{b\max}$, and dividing by the rate of increase in air brake force m_p. If the train comes to a stop before air brake saturation, it instead evaluates to the time until the train stops, as computed by solving Eq. (10) for $v = 0$.

$$\boldsymbol{t_b}(\boldsymbol{v}, \boldsymbol{a}_{\text{b}}) = \mathbf{min}\left(\frac{\boldsymbol{a}_{\boldsymbol{b}\max} - \boldsymbol{a}_{\text{b}}}{\boldsymbol{m_p}}, \frac{(\boldsymbol{b}_{\max} - \boldsymbol{m_s} + \boldsymbol{a}_{\text{b}}) - \left|(\boldsymbol{b}_{\max} - \boldsymbol{m_s} + \boldsymbol{a}_{\text{b}})^{\mathbf{2}} - \mathbf{2}\boldsymbol{m_p}\boldsymbol{v}\right|}{\boldsymbol{m_p}}\right) \tag{11}$$

The distance that the train travels before either stopping or reaching maximum air brake effect is $\int_0^{t_b(v,a_b)} p\ dt = vt_b(v,a_b) + \frac{1}{2}(b_{\max} - m_s + a_b)t_b(v,a_b)^2 + \frac{1}{6}(m_p)t_b(v,a_b)^3$. The velocity of the train after this period of buildup, by Eq. (10), is $v_f = v - (b_{\max} - m_s + a_b)t_b(v,a_b) + \frac{1}{2}m_p t_b(v,a_b)^2$. So, after the brakes finish ramping up, the distance traveled until the train comes to a halt is $\frac{{v_f}^2}{2(b_{\max} - m_s - a_{b\max})}$, using Newton's third equation of motion. If the train stops before finishing brake ramp-up, v_f evaluates to zero, as required. Adding the upper bounds on distance traveled before and after brake ramp-up results in $\text{brakeDist}_a(v,a_b)$ in Eq. (12), an upper bound on braking distance that accounts for the effect of air brakes. While this derivation for controller design is manual, its result will be verified by computer-checked proof in Theorem 3.

$$\begin{aligned}\text{brakeDist}_a(v,a_b) &= vt_b(v,a_b) - \frac{1}{2}(b_{\max} - m_s + a_b)\,t_b(v,a_b)^2 + \frac{1}{6}(m_p)t_b(v,a_b)^3 \\ &+ \frac{\left(v - (b_{max} - m_s + a_b)t_b(v,a_b) + \frac{1}{2}m_p t_b(v,a_b)^2\right)^2}{2(b_{max} - m_s - a_{bmax})} \end{aligned} \tag{12}$$

$$\begin{aligned}\text{stopDist}_{\boldsymbol{a}}(\boldsymbol{p},\boldsymbol{v},\boldsymbol{a_b}) &= \boldsymbol{vT} + \left(\frac{\boldsymbol{a_{max}} + \overline{\boldsymbol{a}}_{\boldsymbol{s}}(\boldsymbol{p}) + \overline{\boldsymbol{a}}_{\boldsymbol{c}}(\boldsymbol{p})}{\boldsymbol{2}}\right)\boldsymbol{T^2} \\ &+ \text{brakeDist}_{\boldsymbol{a}}\left(\boldsymbol{v} + \left(\boldsymbol{a_{max}} + \overline{\boldsymbol{a}}_{\boldsymbol{s}}(\boldsymbol{p}) + \overline{\boldsymbol{a}}_{\boldsymbol{c}}(\boldsymbol{p})\right)\boldsymbol{T}, \boldsymbol{0}\right)\end{aligned}$$

To prove the invariant that after a control cycle of braking, $e - p >$ $\text{brakeDist}_a(v,a_a)$, we must reason about the three dynamically distinct cases: (i) when $\max(a_b, a_{b\max})$ is $a_{b\max}$;

(ii) when it is $\boldsymbol{a}_b$, and $\boldsymbol{t_b}(\boldsymbol{v},\boldsymbol{a}_b)$ evaluates to $(\boldsymbol{a}_{\boldsymbol{b}\max} - \boldsymbol{a}_b)/\boldsymbol{m_p}$; and

(iii) when it is $\boldsymbol{a}_b$ but $\boldsymbol{t_b}(\boldsymbol{v},\boldsymbol{a}_b)$ evaluates to $\frac{(\boldsymbol{b}_{\max} - \boldsymbol{m_s} + \boldsymbol{a}_b) - |(\boldsymbol{b}_{\max} - \boldsymbol{m_s} + \boldsymbol{a}_b)^2 - 2\boldsymbol{m_p v}|}{\boldsymbol{m_p}}$.

The model dynamics are split into these three evolution domains, branching between the three possibilities in a loop to transition between modes freely (Model 3). This does not affect the semantics of evolution: per train dynamics, mode transition happens at most once (from (ii) to (i) or (ii) to(iii)). Unrolling the loop to two iterations, one per mode, suffices to model train behavior. This split formulation simplifies syntactic proofs.

Model 3 Mode-split train dynamics.

$$\text{Dyn.} \left| \begin{array}{ll} 1 & (\quad \{\text{Model 2} \,\&\, a_{b\max} \geq a_b\} \\ 2 & \quad \cup \{\text{Model 2} \,\&\, a_{b\max} \leq a_b \wedge t_a \geq t_p\} \\ 3 & \quad \cup \{\text{Model 2} \,\&\, a_{b\max} \leq a_b \wedge t_a \leq t_p\})^* \end{array} \right.$$

where $t_a = (a_{b\max} - a_b)/m_p$,

$$t_p = \frac{(b_{\max} - m_s + a_b) - |(b_{\max} - m_s + a_b)^2 - 2m_p v|}{m_p}$$

Theorem 3. *The air-brake-exploiting controller guarantees that the train stays within its MA. The* dL *formula below is provable, where* α *is the control loop from Model 1 parameterized with for Eq. (12) for stopDist.*

$$\text{init}_{\boldsymbol{s}}(\text{brakeDist}_{\boldsymbol{a}}) \wedge \text{initAirbrake} \rightarrow \left[\left(\boldsymbol{\alpha}\left(\text{stopDist}_{\boldsymbol{a}}\right); \text{Model } \mathbf{3}\right)^{*}\right]\boldsymbol{e} - \boldsymbol{p} > \mathbf{0}$$

Proof. By proof in KeYmaera X. The high-level idea is to use an outer loop invariant $e - p \geq \text{brakeDist}(v, a_a) \wedge a_{\text{b}} \leq 0 \wedge v \geq 0$ and again split into free driving and braking cases. On braking, we show that the outer loop invariant is maintained in each of the three dynamics modes using an inner loop invariant consisting of four formulas, the most important of which is $e - p > \text{brakeDist}\left(v, \max(a_{\text{b}}, a_{b\max})\right)$.

On free driving, we first show that $\text{brakeDist}_{a0} = \text{brakeDist}_{a}\left(\left(v + \left(a_{\max} + \overline{a}_s(p) + \overline{a}_c(p)\right)T\right), 0\right)$ is truly an upper bound on all reachable $\text{brakeDist}\left(v, \max(a_{\text{b}}, a_{b\max})\right)$ values in the control cycle. Then, holding brakeDist_{a0} constant, we follow a similar proof to show that, in every mode, the increase in p does not exceed the decrease in distance buffer $vT + \left(\frac{a_{\max} + \overline{a}_s(p) + \overline{a}_c(p)}{2}\right)T^2$. The free driving inner loop invariant consists of 14 formulas, most of which state that various upper bound expressions (such as on velocity, grade, and curve) remain upper bounds over the course of the loop. By monotonicity, then, $e - p >$ distance buffer $+ \text{brakeDist}_{a0} \geq \text{brakeDist}\left(v, \max(a_{\text{b}}, a_{b\max})\right)$.

4.5 Conservatively Modeling Air Brakes

The controller of Section 4.4 exploits the force exerted during the gradual ramp-up of air brakes to gain efficiency, resulting in greater mathematical complexity than the preceding controller from Section 4.3. An intermediate controller, used later for validation purposes, accounts for the air brakes only after they have completely ramped up. Ignoring air brakes during ramp-up results in a simpler expression than Section 4.4, but at the cost of less efficiency. This controller is still more efficient than the one from Section 4.3, however.

$$\begin{aligned}
&\text{brakeDist}_{as}(v, a_b) \\
&\qquad = v t_b(v, a_b) - \frac{1}{2}\left(\text{b}_{\max} - m_s\right) t_b(v, a_b)^2 \\
&\qquad\quad + \frac{\left(v - (b_{max} - m_s) t_b(v, a_b)\right)^2}{2(b_{max} - m_s - a_{bmax})} \\
&\text{stopDist}_{\boldsymbol{as}}(\boldsymbol{p}, \boldsymbol{v}, \boldsymbol{a_b}) \\
&\qquad = \boldsymbol{vT} + \left(\frac{\boldsymbol{a_{max}} + \overline{\boldsymbol{a}}_{\boldsymbol{s}}(\boldsymbol{p}) + \overline{\boldsymbol{a}}_{\boldsymbol{c}}(\boldsymbol{p})}{\mathbf{2}}\right)\boldsymbol{T}^{\mathbf{2}} \\
&\qquad + \text{brakeDist}_{\boldsymbol{as}}\left(\boldsymbol{v} + \left(\boldsymbol{a_{max}} + \overline{\boldsymbol{a}}_{\boldsymbol{s}}(\boldsymbol{p}) + \overline{\boldsymbol{a}}_{\boldsymbol{c}}(\boldsymbol{p})\right)\boldsymbol{T}, \mathbf{0}\right)
\end{aligned} \tag{13}$$

The braking distance expression, while fundamentally like brakeDist_a of Eq. (12), omits the terms representing the effect of the air brakes during the buildup phase. The new stopping distance then just uses this new braking distance in the same way as Eq. (12). A controller and safety theorem statement result from these expressions in the usual way.

$$\text{init}_{\boldsymbol{s}}(\text{brakeDist}_{\boldsymbol{as}}) \wedge \text{initAirbrake} \rightarrow \left[\left(\boldsymbol{\alpha}\left(\text{stopDist}_{\boldsymbol{as}}\right); \text{Model } [\mathbf{4}]\right)^{*}\right]\boldsymbol{e} - \boldsymbol{p} > \mathbf{0}$$

4.6 Exploiting Resistance

Exactly accounting for the quadratic dependence of resistance on velocity, as discussed in Section 3.2, leads to an undecidable, transcendental exact solution for stopping distance. The controller must instead use an approximation. Since polynomial arithmetic is decidable, Taylor polynomials are a natural way to obtain decidable approximations. In this section, Taylor approximation is applied to the FRA model, identifying techniques generalizable to verified control for other embedded systems with transcendental dynamics. Railroad systems where rolling resistance is high will benefit particularly from the approximation technique in this section.

The Davis equation implies[3] $\boldsymbol{v' \geq (a_{max} + m_s) + a_1 v + a_2 v^2}$, where slope and curve bounds $\boldsymbol{\overline{a}_s}$ and $\boldsymbol{\overline{a}_c}$ are from Section 4.3. The first-order Taylor polynomial of this expression for velocity is $\boldsymbol{v_0 + \left((a_{\max} + m_s) + a_1 v_0 + a_2 v_0^2\right) t}$. Using this approximation at time $\boldsymbol{T}$, with $\boldsymbol{a_l = a_{\max}}$, as an upper bound for velocity after a period of acceleration, we compute for stopping distance that leverages resistance. While this derivation is manual, its result will be verified by computer-checked proof in Theorem 4.

$$\overline{v}' = (\mathrm{a}_{\max} + m_s) + a_1 v + a_2 v^2$$

$$\mathrm{stopDist}_t(\boldsymbol{v}) = \boldsymbol{vT} + \left(\frac{\boldsymbol{a_{max}} + \overline{\boldsymbol{a}}_s(\boldsymbol{p}) + \overline{\boldsymbol{a}}_c(\boldsymbol{p})}{\boldsymbol{2}}\right)\boldsymbol{T^2} + \mathrm{brakeDist}_{\boldsymbol{a}}\left(\boldsymbol{v} + \left(\overline{\boldsymbol{v}}'\boldsymbol{T}, \boldsymbol{0}\right)\right) \tag{14}$$

Unlike previous stopping distance estimates, this expression is *not* always an upper bound. It uses resistance for the original velocity v_0, which is only a conservative bound when resistance is low enough to permit acceleration. This condition is captured by predicate vbound in Eq. (15).

$$\mathrm{vbound}(\boldsymbol{v}) \equiv \boldsymbol{(a_{\max} + m_s) + a_1 v + a_2 v^2 \geq 0} \tag{15}$$

For the Taylor approximation controller (Model 4), we define the $\mathrm{stopDist}_{tp}$ predicate (16), that unlike previous expressions for stopping distance, returns a truth value. It uses previous definitions $\mathrm{stopDist}_t$ from Eq. (14), vbound (15) to determine when $\mathrm{stopDist}_t$ is applicable, and $\mathrm{stopDist}_b$ from Eq. (8) is used as a fallback.

$$\mathrm{stopDist}_{tp}(p, v, a_b) \equiv e - p > \mathrm{stopDist}_s(p, v, a_b) \vee \left(\mathrm{vbound}(v) \wedge e - p > \mathrm{stopDist}_t(v)\right) \tag{16}$$

Higher order Taylor polynomials permit analogous reasoning. They can provide different, potentially better resistance approximations, possibly covering regions of state space that the first order polynomial does not. In railroad systems, where rolling resistance is a significant effect,

[3] This is a lemma for the proof justified as follows: consider two identical trains on tracks t_1 and t_2, starting with the same velocity. We want to bound the velocity v_1 of the train on t_1. Suppose t_2 is the track with worst case track and grade, and that a train on t_2 (the "ghost train", that we have constructed for the sake of our argument) always accelerates so that $v_2' = (a_{\max} + m_s) + a_1 v + a_2 v^2$. On the other hand, on track t_1, the real train that we require a proof about only obeys the restriction $|a_s| < m_s$. If $v_2 - v_1$ is to become negative, it must cross the boundary where its value is 0. However, whenever $v_1 = v_2$, necessarily, $v_2' \geq v_1'$. This ghost train argument serves a purpose like the "circular dependencies" argument of Section 4.3.1, reasoning about mutually influencing factors one at a time. The ghost train permits us to represent and reason about a transcendental bound on velocity, v_2, derived using slope and curve estimates $\overline{a}_s$ and $\overline{a}_c$.

exploring higher order Taylor polynomials can be a useful way to get more efficient verified controllers. Theorem 4 expresses that the *Taylor polynomial controller* in Model 4 is safe.

Model 4 Taylor polynomial controller and dynamics.

Reset Timer	1	$t := 0;$
	2	$\big((?\texttt{stopDist}_{tp}(p, v, 0);$
Free driving	3	$\quad a_l := *;\ ? - b_{\max} \leq a_l < a_{\max};$
	4	$\quad a_b := 0;\ m_b := 0)$
Braking	5	$\cup\ a_l := -b_{\max}; m_b := m_p\big);$
Dynamics	6	Model 3

Theorem 4. *The Taylor polynomial controller guarantees that the train stays within its MA. The* dL *formula below is provable.*

$$\text{init}_{\boldsymbol{s}}(\text{brakeDist}_{\boldsymbol{a}}) \wedge \text{initAirbrake} \rightarrow [(\text{Model } [\mathbf{4}])^*]\boldsymbol{e - p > 0}$$

Proof. By proof in KeYmaera X. We start by showing that the loop invariant from Theorem 3 is maintained (using dL rule loop). The case where the train is braking proceeds like Theorem 3. When the train is accelerating, we need to show that the controller has insisted on a sufficient distance margin (i.e., stopping distance), so that even after one period, the train has enough space to stop. As the Taylor polynomial computes the stopping distance, we must prove that it is indeed an upper bound. We use a monotonicity argument by introducing an auxiliary variable that represents a "ghost" train, perpetually traveling down worst possible slope $\overline{a}_s$ and curve $\overline{a}_c$. This isolates slope and curve from the effect of resistance, breaking interdependence. We derive the Taylor polynomial result on the ghost train, show that it goes no slower than the real train, and that consequently the Taylor polynomial result must hold for the real train. Other elements of the proof remain like Theorem 1.

4.7 Automated Synthesis of Control Conditions

Safe control envelopes, as developed in the previous sections, are nondeterministic programs whose every execution is safe. In contrast with controllers, control envelopes define entire families of controllers to allow control actions under as many circumstances as possible, if they maintain the safety of the hybrid system. Safe control envelopes allow the verification of abstractions of control systems, isolating the parts relevant to the safety feature of interest, without involving the full complexity of a specific control implementation. The full control system is then monitored for adherence to the safe control envelope at runtime. The control envelope approach allows a single verification result to apply to multiple specialized control implementations, optimized for different objectives. It puts industrial controllers that are too complex to verify directly within the reach of verification, because a control envelope only needs to model the safety-critical aspects of the controller. Control envelopes also enable applications like justified speculative control (Fulton & Platzer, Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning. AAAI 2018: 6485-6492, 2018), where machine-learning-based agents control safety-critical systems safeguarded within a

verified control envelope. Control envelopes also enable generating safety reward components, as detailed in Section 9.

Control envelope design is challenging. Engineers are usually able to specify the shape of a model and list the possible control actions by translating client specifications, which is crucial for the fidelity of the resulting model. But identifying the exact control conditions required for safety is a much more difficult problem that requires design insights, creativity, and knowledge of control theory (the previous sections developed such control conditions manually). Most initial system designs are incorrect and must be fixed before verification succeeds. Fully rigorous justification of the safety of the control conditions requires full verification of the resulting controller in the hybrid systems model. In (Kabra, et al., 2024; Kabra, et al., 2023), a synthesis technique is presented that addresses this difficult problem by filling in the holes of a hybrid systems model to identify a correct-by-construction control envelope that is as permissive as possible.

4.8 Kinematic Train Model Proofs

A proof for the model (Brosseau & Ede, 2009) derives from the proof for the abstract mathematical models, e.g. Theorem 4, by uniform substitution (Platzer, 2017), which replaces abstract function symbols with specific terms using the correspondence in Section 4.3. Model 5 lists the model resulting from substitution.

Model 5 Train controller in kinematic motion model.

Reset Timer	1	$t := 0;$
	2	$\big($ $?\texttt{stopDist}_{tp}(p, v, 0);$
Free Driving	3	$a_l := *;\ ?{-b_{\max}} \le a_l < a_{\max};$
	4	$a_b := 0;\ m_b := 0$
Braking	5	$\cup\ a_l := -b_{\max};\ m_b := m_p\big);$
	6	$($ $\{\text{dyn}\ \&\ a_{b\max} \ge a_b\}$
	7	$\cup\ \{\text{dyn}\ \&\ a_{b\max} \le a_b$
Mode Split	8	$\wedge\ (a_{b\max} - a_b)/m_p \ge t_p\}$
	9	$\cup\ \{\text{dyn}\ \&\ a_{b\max} \le a_b$
	10	$\wedge\ (a_{b\max} - a_b)/m_p \le t_p\})^*$

$$\text{with dyn} \equiv \{p' = v, a_b' = m_b, t' = 1, v' = a_l + \max(a_b, a_{b\max}) - \frac{A_G w \cdot \mathit{grade}(p)}{m_T} - \frac{A_C w \cdot \mathit{curve}(p)}{m_T} - \frac{C_R w v + D_R v^2}{m_T}\ \&\ t \le T \wedge v \ge 0\}$$

Corollary 5 (Kinematic train model is safe). *The train controller for the FRA model never overshoots the EOA, i.e., the following formula is provable with* $b_{\max} = \frac{F_B}{m_T} + \frac{A_R w + B_R n}{m_T}$ *and* $a_{\max} = \frac{F_L}{m_T} - \frac{A_R w + B_R n}{m_T}$*:*

$$\text{init}_s(\text{brakeDist}_a) \wedge \text{initAirbrake} \rightarrow [(Model\ 5)^*]e - p > 0$$

Proof. By uniform substitution from Theorem 4, using the substitutions σ below:

$$\sigma = \begin{cases} \boldsymbol{a_{max}} \mapsto \dfrac{\boldsymbol{F_L}}{\boldsymbol{m_T}} - \dfrac{\boldsymbol{A_R w + B_R n}}{\boldsymbol{m_T}} \\ \boldsymbol{a_s(p)} \mapsto -\dfrac{\boldsymbol{A_G w \cdot grade(p)}}{\boldsymbol{m_T}} \\ \boldsymbol{a_r(v)} \mapsto -\dfrac{\boldsymbol{C_R w v + D_R v^2}}{\boldsymbol{m_T}} \\ \boldsymbol{b_{max}} \mapsto \dfrac{\boldsymbol{F_B}}{\boldsymbol{m_T}} + \dfrac{\boldsymbol{A_R w + B_R n}}{\boldsymbol{m_T}} \\ \boldsymbol{a_{c(p)}} \mapsto -\dfrac{\boldsymbol{A_C w \cdot curve(p)}}{\boldsymbol{m_T}} \end{cases}$$

4.9 Evaluation

For validation, ModelPlex (Mitsch & Platzer, 2014) is used to derive a controller monitor that measures the safety margin in decisions of previous brake enforcement controllers (Brosseau, et al., 2013; Brosseau & Ede, 2009) and the verified control. That way, we measure if, and how well, the verified train controllers and existing controllers agree on a numeric simulation of the train kinematics of Eq. (2) to assess the safety of those existing systems and the efficiency of the model. Existing brake enforcement controllers brake to a full stop once engaged.

The ModelPlex monitor computes a robustness measure indicating how close a decision is to losing the safety proof. When the robustness measure is positive, the decision is guaranteed to remain provably safe so that the system enjoys the safety proof of the verified model. When it is negative, emergency brakes should be applied for safety reasons. The ModelPlex controller monitor follows the structure of the verified model when it computes robustness. For example, a monitor for Lines 2-3 of Model 5 describes their effect with the formula $\text{stopDist}_{\boldsymbol{tp}}(\boldsymbol{p}, \boldsymbol{v}, \boldsymbol{0}) \wedge -\boldsymbol{b}_{\max} \leq \boldsymbol{a_l} < \boldsymbol{a}_{\max}$ which translates to the robustness measure below:

$$\min\Big(\max\Big(\min\big(e - p - \text{stopDist}_t(v), (a_{max} + m_s) + a_1 v + a_2 v^2\big), e - p - \text{stopDist}_{a(p,v,0)}\Big), a_l + b_{max}, a_{max} - a_l\Big)$$

The most important elements of the full Model 5 monitor are:

- in free driving (when stopDist_{tp} is satisfied) it combines remaining position margin (the larger of Taylor margin $e - p - \text{stopDist}_t$ when vbound(v), or fallback margin $e - p - \text{stopDist}_a$) with acceleration choice robustness ($\min(a_l + b_{\max}, a_{\max} - a_l)$ from control decision $a := *; ?-b_{max} \leq a_l < a_{max}$) and speed robustness ($v$ from evolution domain ... $\& v \geq 0$);
- during braking, which is always allowed, it measures speed robustness (v per evolution domain constraint).

Because ModelPlex's robustness measure combines multiple quantities of incompatible units, there is no direct interpretation of its magnitude, only of its sign.

For validation, the train model of Eq. 2 is implemented in Python by numerical integration, instantiating the model parameters per FRA model (Brosseau, et al., 2013; Brosseau & Ede, 2009). These parameter values are estimated from train test runs and standards (Brosseau J. , Ede, Pate, Wiley, & Drapa, 2013) and require careful consideration of their units (Table 1).

The evaluation compares start braking and stopping points of trains, highlighting braking performance in terms of overshoot (safety risk) and undershoot (performance objective of a maximum undershoot of $\mathbf{1000}$ft (Brosseau & Ede, 2009) of the EOA. We follow (Brosseau J. , Ede, Pate, Wiley, & Drapa, 2013) and implement the baseline controllers using numeric forward Euler integration to simulate the model to determine the stopping distance. The verified controllers neither use numeric integration nor include the dynamic model, but instead decide based on the stopping distance over approximation $\text{stopDist}_{\boldsymbol{tp}}$ of Eq. (16).

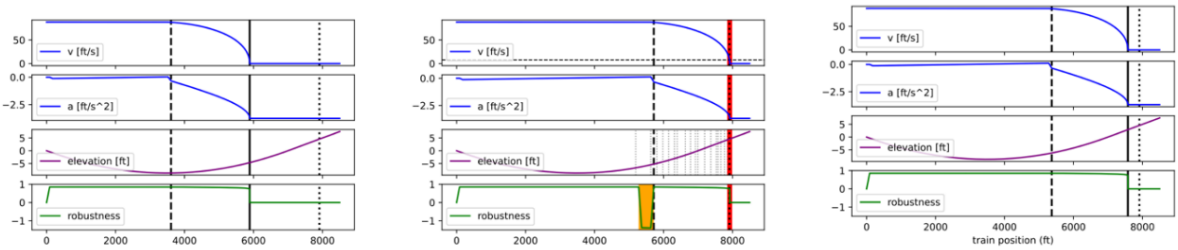
Table 1. FRA train parameter instantiation example

Param.	Value	Unit	Description
A_R	0.6	$lbf/_{ton}$	(constant) weight coefficient
B_R	20	$lbf/_{axle}$	(constant) train-size coefficient
C_R	0.01	$lbf/_{ton \cdot mph}$	(linear) speed coefficient
D_R	0.07	$lbf/_{mph^2}$	(quadratic) aerodynamic coefficient
	0.294		(for loaded cars)
A_G	20	$lbf/_{ton}$	(constant) weight coefficient

The most interesting train behavior arises from the subtle interplay between air pressure propagation, aerodynamic/roll resistance, and acceleration/deceleration due to slope. It peaks on crests that change gradient from uphill to downhill and in troughs that change gradient from downhill to uphill. When calculating stopping distance, numerical integration in the baseline enforcement algorithms discretizes train speed and position to calculate forces, which overestimates resistance while simultaneously underestimating available brake force. Acceleration/deceleration due to slope is even more subtle as it depends on the position of the train along the slope (e.g., on a crest changing from uphill to downhill, deceleration on the uphill segment is overestimated until the train passes the top, afterwards acceleration is underestimated). These effects do *not* balance out and thus make numerical integration errors unreliable and hard to predict. Moreover, changing the integration step size shifts how distance estimates are biased toward undershoot or overshoot (e.g., in typical configurations, brake ramp-up is the dominating influence on stopping distance, and so larger integration step sizes bias toward undershoot). As a result, for any given configuration of numerical integration in enforcement algorithms, scenarios can be constructed where the numerical integration underestimates stopping distance and train enforcement exhibits unsafe behavior. The formal models and proofs design *provably correct* stopping distance over approximations instead of using numerical integration and are, therefore, not subject to these intricate safety tradeoffs.

The first validation in Figure 2 follows (Brosseau & Ede, 2009) with a train configuration of a medium (75 cars), loaded (car weight $\mathbf{286}$klbf), mixed freight train traveling at initial speed $\mathbf{60}$mph in a trough, with train starting position at $\mathbf{0}$ft. The trough is configured with uniform rate of change from $\mathbf{0.5}\%$ downhill to $-\mathbf{0.5}\%$ uphill between positions $\mathbf{0}$ft and $\mathbf{7000}$ft. Locomotive

tractive effort and locomotive braking compensate for speed loss and gain due to gradient. Air brakes engage only for the full braking maneuver when attempting to not exceed the MA. The monitors and controllers are configured to match the trough maximum uphill/downhill gradient of $\mathbf{0.5}\%$ for slope estimation. A sweep of MA endpoints in the range $\mathbf{4000}$ft–$\mathbf{8000}$ft with $\mathbf{10}$ft steps identifies challenging configurations. The base enforcement algorithm (Brosseau & Ede, 2009) underestimates stopping distance for 19 of the 400 endpoints. Figure 2 compares the base enforcement algorithm to the verified controller on one of these challenging points at $\mathbf{7910}$ft, past the maximum uphill slope of the trough. The base enforcement algorithm in Figure 2(a) uses numeric integration to determine the stopping distance, and adds a generous constant fudge factor plus speed-dependent safety offset to result in the train initiating braking at $\mathbf{3609}$ft. The maneuver finishes at $\mathbf{5889}$ft, stopping $\mathbf{2021}$ft short of the EOA. In Figure 2(c), the verified controller does *not* use any such offsets. It initiates braking much later at position $\mathbf{5368}$ft and stops at $\mathbf{7578}$ft (significantly closer to the MA endpoint, limiting undershoot to $\mathbf{332}$ft).



(a) Base enforcement algorithm [1] initiates braking at 3609ft and stops at position 5889ft with an undershoot of 2021ft, which exceeds the 1000ft maximum undershoot performance objective [1].

(b) Base enforcement algorithm [1] without offset overshoots by 9ft ϟ. Light-gray dotted lines mark other challenging movement authority endpoints in the range 4000ft–8000ft that would also be violated.

(c) Verified controller (Model 5) limits undershoot to 332ft.

Figure 2. Comparison of start braking (dashed vertical line) and stopping points (solid vertical lines) in a trough from 0.5% downhill at start to −0.5% uphill at 7000ft, with EOA (dotted vertical line) at 7910ft.

Figure 2(b) removes the safety offset from the base enforcement algorithm in an experiment illustrating the subtle interplay between forces and their safety consequences. Even though the uphill segment of the trough helps reduce stopping distance, numeric integration overestimates this effect and initiates braking too late, which the monitor detects at $\mathbf{5368}$ft. Should the train ignore the warning, it overshoots by $\mathbf{9}$ft (highlighted in red[4]) with a remaining speed of $\mathbf{7.8}\,\frac{\text{ft}}{\text{s}}$ when passing the desired stopping point. Initiating fallback control $\boldsymbol{a_l} = -\boldsymbol{b}_{\max}$ at the monitor violation would have kept the train from overshooting the MA.

4.10 Stopping Behavior on Crests

An assumption in the brake enforcement algorithms and thus an initial condition in the proof is that train locomotives are not underpowered; their tractive effort is enough to overcome

[4] The controller monitor subsequently no longer flags a violation, because the base enforcement algorithm then applies the correct decision of maximum braking (agreeing with the model). But it does so too late, as the earlier monitor warning was ignored.

maximum uphill slope and stay stopped on the maximum downhill slope. Figure 3 illustrates how an underpowered, initially stopped train starts rolling downhill until the air brakes build enough deceleration to stop the train. This configuration, violating initial conditions, is unsafe to start in the first place.

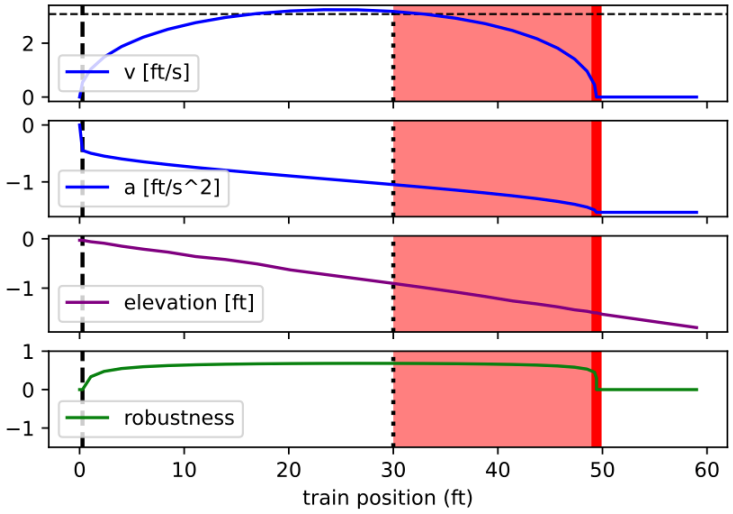
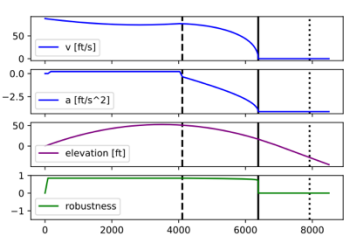


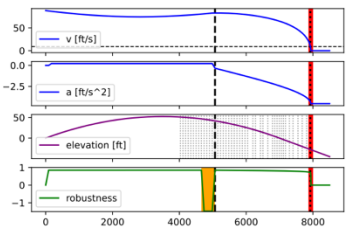
Figure 3. Initial conditions of proof (Corollary 5) not satisfied: an underpowered train on a 3% downhill slope needs air brakes to stay stopped; the stopping point is closer than the rolling distance during brake ramp-up.

Underpowered locomotives are especially challenging on a crest where (full) tractive effort is needed to limit the speed loss on the uphill slope and regain desired speed on the downhill segment, but air brakes are needed to stay stopped. Figure 4 compares the behavior of the base enforcement algorithms and the verified algorithm with underpowered locomotives on a crest with MA endpoint at **7920**ft. In this experiment, the monitors and controllers are configured for a maximum uphill/downhill gradient of **3**% for slope estimation.

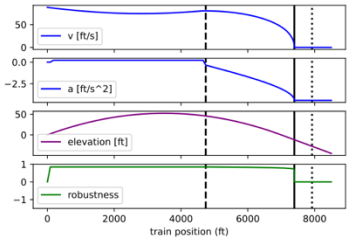
Numeric integration underestimates stopping distance in several configurations, whereas the verified controller correctly identifies the need to engage air brakes in time while simultaneously avoiding the inefficiencies of fudge factors.



(a) Base enforcement algorithm [1] undershoots by 1539ft.



(b) Base enforcement algorithm [1] without offset overshoots by 14ft ϟ. Light-gray dotted lines mark other movement authority endpoints in the range 4000ft–8000ft that the algorithm would also overshoot.



(c) Verified controller (Model 5) limits undershoot to 528ft.

Figure 4. Initial conditions of proof (Corollary 5) not satisfied: an underpowered train on a crest from 3% uphill to −3% downhill slows down uphill despite full tractive effort and regains speed on the downhill segment but needs air brakes to stay stopped; in this scenario, the MA ends past the rolling distance during brake ramp-up.

5. Testing-based Approach to Model Parametrization with TOES

The goal of the test-based model parametrization is to check whether there exist parametrizations of the symbolic formal models such that the stopping distance predictions of the verified controllers closely resemble the stopping distances of a real-world train controller (stopping distance refers to the distance a train needs to fully stop from the point of brake application). A parameterized formal model is called a model instance. To demonstrate the validation approach, the parametrization of (Brosseau & Ede, 2009) (Brosseau et al., 2013) is used. Railroad companies would perform similar validation with their own form of parametrization.

Real-world train controllers, in operation by different vendors, are based on different implementations. To not bias the validation to a specific vendor, the team chose the industry standard train simulator TOES, a physics train simulator from the Transportation Technology Center Inc. (TTCI), which has been validated against real world behavior. TOES can run test scenarios with different train and track configurations. The train scenarios capture a large range of train consists, speed, and grades. Train consists vary by the type of train (e.g., unit, intermodal, and manifest), the type of cars trailing (e.g., boxed, multi-level, tanks, etc.) as well as their weight and power distribution. With the large number of configurations, a large variation of brake distances were observed across simulation runs. Furthermore, even simulations started with equal initial train and track configuration lead to a considerable variation in braking distances. This is caused by silent variation of brake-relevant details internally in the simulator. Silent variations, summarized in (Brosseau et al., 2013, Appendix D), are not exposed outside of the simulator, but used internally in the simulator to approximate the deviation between initial (assumed) train and track configurations and actual parameters. In this validation, no attempt was made to characterize this uncertainty in the parametrization but all tests were run with model instances based on the parametrization of (Brosseau & Ede, 2009) (Brosseau et al., 2013) with the initial (assumed) train and track configurations. Despite safety proofs for all formal models, it is therefore expected to see overshoot as well as undershoot depending on how the simulator internally varies the starting configuration.

A model instance passes validation if the difference in stopping distance between the model instance and the simulated stopping distance is within a defined margin for all train scenarios. Since no attempt was made to characterize the internal variation of the simulator, there is no single model instance based on the parametrization of (Brosseau & Ede, 2009) (Brosseau et al., 2013) that meets the stopping distance difference margin requirement uniformly well across all simulations. Thus, a more fine-grained parametrization for train configurations and even single trains is needed to accommodate for the differences. However, due to the silent internal variation in the simulator, this approach becomes infeasible, i.e., parameters identified for a 100-car train in one simulation do not hold for the next time the simulation is run, as different parameters for the 100-car train would be identified.

5.1 Validation Setup and Approach

The validation approach focuses on comparing the stopping distances of model instances to those of the TOES train simulator. A test framework was developed which runs the defined test scenarios on the model instances and compares the predicted results of stopping distances with the simulator-calculated stopping distances. The results are depicted in plots for a manual analysis as described in Section 7.1.4.

5.1.1 Parametrized Model Under Validation

For validation, the team created instances of the stopping distance prediction functions Eq. (4) ("Baseline Model"), Eq. (13) ("Delaybrake Model"), and Eq. (12) ("Airbrake Model") of the formal models. The purpose of the model instances is to predict stopping distance while training a reinforcement learning agent to monitor safety and determine the rewards. The formal models and their stopping distance prediction functions are fully symbolic; they symbolically encode crucial relationships between train weight, maximum brake force, curve resistance, rolling resistance, gravity, and other parameters, as well as their influence on stopping distance, but the concrete values of these constants are left unspecified. To use such a symbolic formal model on a concrete train or to compare with a concrete simulation run, concrete values for all the model parameters must be chosen; the formal model with concrete values for the parameters is called a model instance. Instantiating a formal model is an important task in using the verification results in real trains. If model parameter values are chosen poorly, the model instance may predict stopping distances too conservatively (e.g., if brake parameters indicate that brakes are worse than the true train brakes) or too aggressively (e.g., if brakes are parameterized substantially better than the true train brakes). Whether or not a model instance fits reality can be analyzed by runtime monitoring (Mitsch & Platzer, 2014), but a poorly parameterized model instance results in monitors that interfere often. Validating whether and in which concrete scenarios a model instance resembles true real-world stopping distances is crucial in identifying how well a parameterization fits reality.

5.1.2 Train Simulator and Test Scenarios

The TOES simulator is based on a physics model representing the behavior of a train. Since the simulator does not underlay the restrictions of an embedded controller, its fidelity of considering physical train characteristics is more detailed than an actual train controller onboard unit. TOES includes a predefined list of test scenarios with their expected simulation results of the stopping distance. A total of 400 train scenarios that differ in their train and track configurations are defined. A subset of these scenarios were tested, grouped into three main buckets of train configurations of intermodal (70 configurations), manifest 114 configurations), and unit (72 configurations) trains. A test scenario is a combination of a specific (i) train consist, (ii) track topography, (iii) initial location, (iv) initial speed, (v) target location, and (vi) target speed. The result of a simulated scenario provides the expected stopping distance. All original data was collected by TOES, in batches with each data point having a batch name and batch line (their combination uniquely identifies a simulation run). A batch represents a collection of similar train configurations including the type of the train and its consist.

Structure of the Simulation Data. For replay without a TOES instance, TOES simulation data of all scenarios were collected in a database running Microsoft SQL Server 11. Table 2 shows the simulation scenario content, Table 3 summarizes the track characteristics, and Table 4 summarizes the characteristics of a train consist. All track profiles have a constant linear characteristic, i.e., their altitude is proportional increasing or decreasing over distance. However, steeper downhill slopes start with a smaller gradient. The smaller gradient is a 0.5 percent decrease for the first 50k ft. At the tipping point of 50k ft the gradient changes into a higher value, as illustrated in Figure 5.

Table 2. Simulation scenario characteristics

Column name	Data type	Content
BatchLine	int	Simulation scenario identifier, primary key
BatchName	varchar(20)	
Consist	varchar(20)	Consist identifier → TrainName in Table 4
TrackFile	varchar(20)	Track geometry identifer → TrackName in Table 3
TrainHandlingFile	varchar(20)	
CalcPropTime	float	
CalcWeightCorrection	float	
CalcBrakeEff	float	
TargetStopPositionFeet	float	Position of the brake target in feet along the track
InitTrainVelocityMph	float	Initial speed of the train in miles per hour
InitTrainPositionFeet	float	Initial position of the front end of the train in feet
AccelProfile	int	
Direction	varchar(20)	
Record	varchar(20)	
Notch	tinyint	
CalcTimeToApplySec	float	
CalcTrainWeightTons	float	
CalcBrakeForce	float	
Throttle	varchar(12)	
TargetStopSpeedMph	float	Speed limit at target location (always 0 here)
BrakeInitFile	int	

Table 3. Track characteristics

Column name	Data type	Content
TrackName	varchar(32)	Track identifier, primary key
Marker	varchar(7)	Type of this point, one of: **TRKBGN** first point of the track **ELVATN** any point within the track **TRKEND** last point of the track
Footage	numeric(15,3)	One-dimensional coordinate along the track, in feet
Curve	numeric(9,2)	Curvature of the track at this point
Superelevation	numeric(9,3)	Superelevation of the track at this point
ElevationFt	numeric(9,3)	Elevation of the track at this point in feet
GradePercent	numeric(6,3)	Grade of the track at this point in percent
SpeedLimitMph	numeric(5,2)	Speed limit at this location in miles per hour
Lubrication	bit	Flag to indicate track lubrication
Milepost	varchar(10)	Milepost name
Station	varchar(20)	Station name

Table 4. Characteristics of a train consist

Column name	Data type	Content
TrainName	nchar(20)	Name of the train, unique key
Orientation	tinyint	
TrainType	int	Type of train
TrailingTons	int	Total weight of cars in the train in short tons
CarsNoBrakes	int	Number of cars in the train with disabled brakes
Axles	int	Number of axles of cars in the train
TrainLength	int	Total length of the train in feet
Loads	int	Number of loaded cars in the train
Empties	int	Number of empty cars in the train
CarBrakeForce	int	
EmergencyBackupBrake	tinyint	
EBB	tinyint	
Locomotives	int	Number of locomotives in the train
Sequence	nchar(20)	

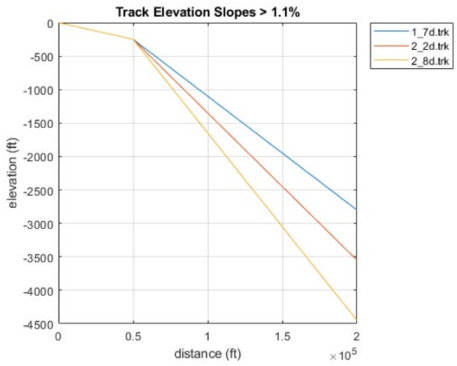


Figure 5. Track profile – steep slope.

5.1.3 Validation Approach

Figure 6 depicts the overall view of the test approach. MATLAB was used as the test execution engine, which creates model instances by feeding the configuration parameters of each simulation run to the formal model to create model instances, and then uses the model instances to calculate predicted stopping distances. The model instance stopping distances are then visualized in comparison to the expected stopping distances recorded from the TOES simulator. The scenarios and their simulated TOES stopping distances were captured in a database as described in Section 9.2.

For the test execution engine to create model instances, the stopping distance functions of the formal models are translated first into Mathematica (through the Mathematica export of the theorem prover KeYmaera X (Fulton et al., 2015), and from there into a MATLAB representation that can be executed by the validation framework. For MATLAB export, the Mathematica package "ToMatlab" was used, which is available in the Wolfram Library Archive Package ToMatlab, latest revision 1999 − 04 − 24. The stopping distance exported from the

model instance and the validation framework are aligned by feeding the same simulation configuration data as parameters to the model and the simulator.

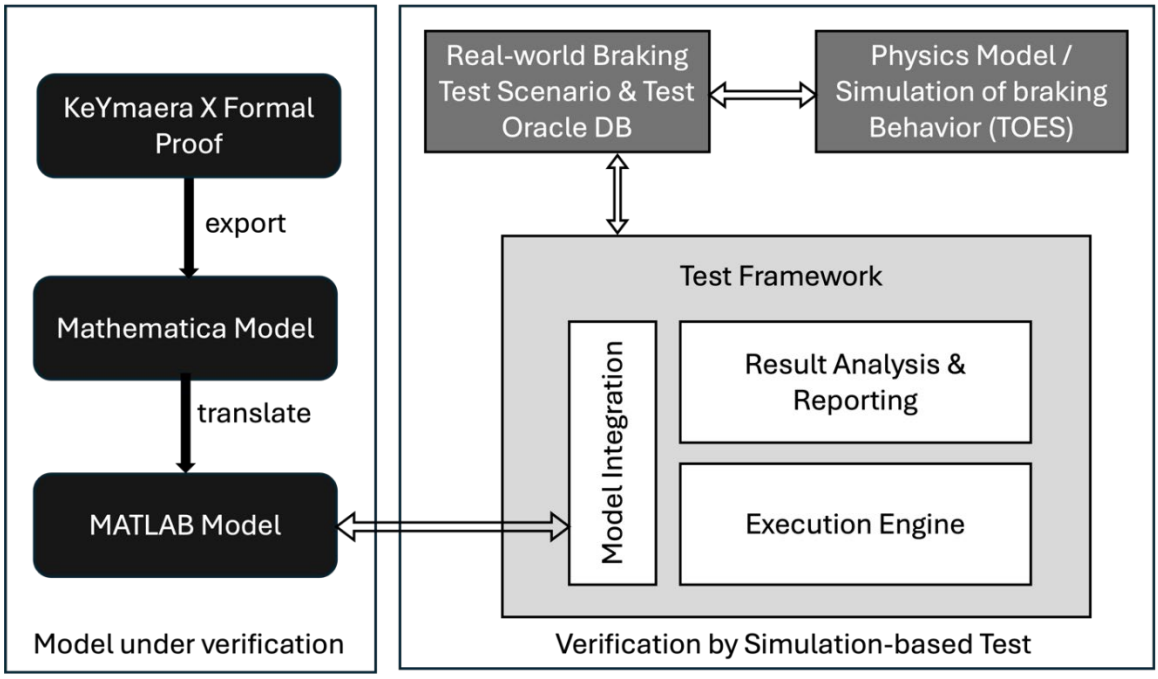


Figure 6. Test setup overview.

In summary, the overall validation flow can be described by four steps providing the two inputs consisting of the MATLAB model and the TOES simulation reference data from TTCI.

1. Read in simulation run configurations and expected stopping distances from database
2. Use simulation run configuration to parameterize formal model and create model instance under test
3. Compute stopping distance using model instance
4. Plot model instance stopping distance in comparison to expected TOES stopping distance

5.1.4 Evaluation Criteria and Analysis

The braking model should satisfy two evaluation criteria. One is to meet its safety requirement and the other to meet its performance requirement. The safety requirement requests that 95.5 percent of all runs need to stop before the stopping target. Any overshoot of the target is counted as an unsafe scenario (see Figure 7). The performance measures are limited by the actual speed of the train. Faster trains are allowed to stop further away from the target while slower trains should get closer to the actual stopping target. The two performance measures are:

- Up to 30mph: The train needs to stop at most 500ft before target
- Above 30mph: The threshold to stop before the target is 1200ft

Figure 7. Illustration of summary classification of stopping distance.

The purpose of the result analysis and reporting module of the test framework (see Figure 6) is to evaluate the results of the execution runs against the above criteria. The evaluation and analysis are provided by three main plots generated:

1. Overall Stopping Distances (scatter plot): This is a comparison of median stopping distances be- tween model instances and TOES over each individual train consist. It detects when prediction from the model instance (parametrization) and simulation disagree and provides an overview and tendencies of the scenarios. The median of each consist is represented by a circle. The variations of TOES stopping distances (coming from the Monte-Carlo Simulation) are illustrated with horizontal error bars. Two "Overall Stopping Distance" plots are made which differ by their initial velocity, represented in different colors: the green points represent consists of speed less than 30mph, whereas red points represent consists of speed with at least 30mph. Thus, each plot can be reviewed against the overshoot/undershoot requirements above, which are sorted into the same speed categories.
2. Stop Distance Deviation (histogram): The diagram depicts the stopping distance deviation probability. The diagram highlights the distribution of stopping distance by their deviation.
3. Performance Evaluation (pie chart): The performance evaluation chart visualizes how many stopping distances of the model instances satisfy the performance requirements. We differentiate between "excessive undershoot," "good," and "overshoot" cases. It shows how close chosen parameters of the model instance are predicting the expected TOES stopping distance.

5.2 Validation Process and Results

The validation was performed over three major development releases of the formal model which are discussed in Section 7.2.1.

Baseline Model Eq. (4). The baseline model assumes full brake force is available at the time of the brake engagement. Although known to be physically unrealistic, the main target of this step is to align model and simulator quantities and their units. The baseline model instances were prepared using the substitutions and unit conversions below:

$$\sigma = \begin{cases} v\,[\text{ft/s}] \mapsto v_{\text{mph}} \frac{5280}{3600} & \text{with } v_{\text{mph}} \text{ in [mph]} \\ a_{\max}\,[\text{ft/s}^2] \mapsto \frac{F_L}{m_T} - \frac{A_R w + B_R n}{m_T} & \text{with } w \text{ in [tons]}, n \text{ number of axles}, \\ & \quad F_L = 0\,[\text{lbf}], m_T = \frac{2000w}{g}\,[\text{lbm}], g = 32.2\,[\text{ft/s}^2] \\ & \quad A_R = 0.6\,[\text{lbf/ton}], B_R = 20\,[\text{lbf/axle}] \\ b_{\max}\,[\text{ft/s}^2] \mapsto \frac{F_B}{m_T} + \frac{A_R w + B_R n}{m_T} & \text{with } F_B \text{ in [lbf]}, w, A_R, B_R, m_T, g \text{ as above} \\ m_s\,[\text{ft/s}^2] \mapsto -\frac{A_g grade \cdot g}{2000} = -\frac{A_g grade \cdot w}{m_T} & \text{with } \textit{grade} \text{ percentage (+uphill, -downhill)}, \\ & \quad A_g = 20\,[\text{lbf/ton}], g, w, m_T \text{ as above} \\ T\,[\text{s}] \mapsto 1\,[\text{s}] \end{cases}$$

Delaybrake Model Eq. (13). The delaybrake model improves the baseline model by deferring air brake application of a_{bmax} for the entire brake pressure propagation time, while still using an instantaneous component F_B to compensate for grade. The delay overapproximates brake force ramp-up, comparable with a delayed step function. The delaybrake model instances were prepared using the following substitutions and unit conversions:

$$\sigma = \begin{cases} v\,[\text{ft/s}] \mapsto & \text{see Baseline model} \\ t_b(v, a_b)\,[\text{s}] \mapsto \min(\frac{v}{a_b - m_s}, t_{\text{appl}}) & \text{with } a_b = \frac{F_B + A_r w + B_r n}{m_T}\,[\text{ft/s}^2], \\ & \quad t_{\text{appl}} = 12.22 + 0.0156L - 0.000000278L^2\,[\text{s}], \\ & \quad L \text{ in [ft]} \\ a_{\max}\,[\text{ft/s}^2] \mapsto & \text{see Baseline model} \\ b_{\max}\,[\text{ft/s}^2] \mapsto & \text{see Baseline model} \\ \overline{a}_s(p)\,[\text{ft/s}^2] \mapsto -\frac{A_g grade(p) \cdot g}{2000} = -\frac{A_g grade(p) \cdot w}{m_T} & \text{with } \textit{grade}(p) \text{ percentage (+uphill, -downhill)}, \\ & \quad p \text{ track position in [ft]} \\ & \quad A_g, g, w, m_T \text{ see Baseline model} \\ m_s\,[\text{ft/s}^2] \mapsto \overline{a}_s(p) & \text{with max. grade for } \textit{grade}(p) \\ \overline{a}_c(p)\,[\text{ft/s}^2] \mapsto 0\,[\text{ft/s}^2] & \text{with } p \text{ as above} \\ T\,[\text{s}] \mapsto 1\,[\text{s}] \end{cases}$$

Airbrake Model Eq. (12). The airbrake model considers brake force ramp-up due to air brake pressure propagation, modeled with a linear ramp-up function until brake force saturation. The airbrake model instances were prepared using the following substitutions and unit conversions:

$$\sigma = \begin{cases} v\,[\text{ft/s}] \mapsto & \text{see Baseline model} \\ a_{\max}\,[\text{ft/s}^2] \mapsto & \text{see Baseline model} \\ b_{\max}\,[\text{ft/s}^2] \mapsto & \text{see Baseline model} \\ \overline{a}_s(p)\,[\text{ft/s}^2] \mapsto & \text{see Delaybrake model} \\ m_s\,[\text{ft/s}^2] \mapsto & \text{see Delaybrake model} \\ \overline{a}_c(p)\,[\text{ft/s}^2] \mapsto 0\,[\text{ft/s}^2] & \\ a_b\,[\text{ft/s}^2] \mapsto 0\,[\text{ft/s}^2] & \\ m_p\,[\text{ft/s}^3] \mapsto \frac{a_{b\max}}{t_{\text{appl}}} & \text{with } a_{b\max} = \frac{F_{B,BCP}}{m_T}, \\ & \quad F_{B,BCP} \text{ maximum air brake force in [lbf]}, \\ & \quad t_{\text{appl}} = 12.22 + 0.0156L - 0.000000278L^2\,[\text{s}], \\ & \quad L \text{ train length in [ft]} \\ T\,[\text{s}] \mapsto 1\,[\text{s}] & \end{cases}$$

Each model release was validated against the same set of test scenarios. The validation results are discussed below.

5.2.1 Analysis and Results

V1: Baseline Model. The diagrams in Figure 8 show the detailed results in stopping distance for the baseline model. The baseline stopping distance function Eq. (4) is an optimistic model by assuming that the total brake force is immediately available without brake force buildup due to air pressure propagation. This assumption is reflected in the far shorter stopping distances compared to the TOES simulation results. As expected, the deviation from TOES calculated stopping distances are widely distributed, with deviation up to 8000ft. The interesting insight of Figure 8(a) and Figure 8(b) is the extent to which the simulator silently varies parameters across simulation runs in a single train consist despite being started with the same starting configurations; the horizontal error bars in the plot represent the variation in simulator output around the median of stopping distances of a consist starting configuration. Due to the late brake initiation caused by the instantaneously available brake force, the test runs result in an overshoot, as should in Figure 9(b).

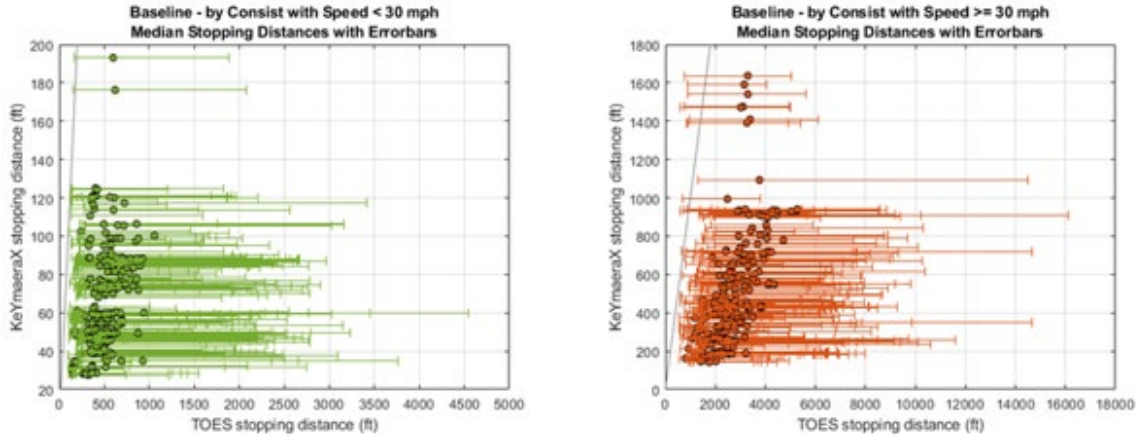


Figure 8. Stopping distances by consist (one circle per consist, horizontal position is media TOES distance, vertical is median baseline model distance, error bars for silent variation).

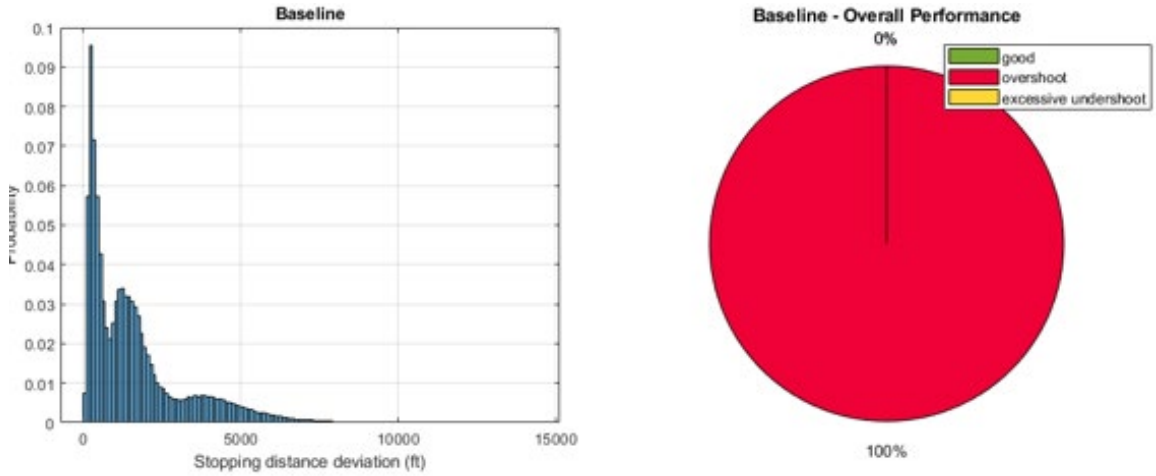


Figure 9. Baseline model instance validation summary.

Nevertheless, the baseline model indicates that the parameters are translated correctly to model instances as the trend of model instance predictions follows the simulator trend, meaning that heavier trains have longer stopping distances in both models. The outlier examples where the simulator reports shorter stopping distance than that which instant brakes achieve can be due to the simulator using additional dynamic brakes of the locomotives for short trains, while the baseline stopping distance Eq. (4) was configured with only air brake force (dynamic brakes can be added by instantiating Eq. (4) with the sum of air brake force and dynamic brake force).

Comparing longer trains in Figure 10 with more than a hundred cars does reveal that the baseline model indeed has its main impact to the stopping distance of longer trains, which was expected. In conclusion, the baseline stopping distance Eq. (4) with its simplified brake model is a useful first step to guide parameterization. Additional brake details improve similarity between formal model and simulator, as validated next.

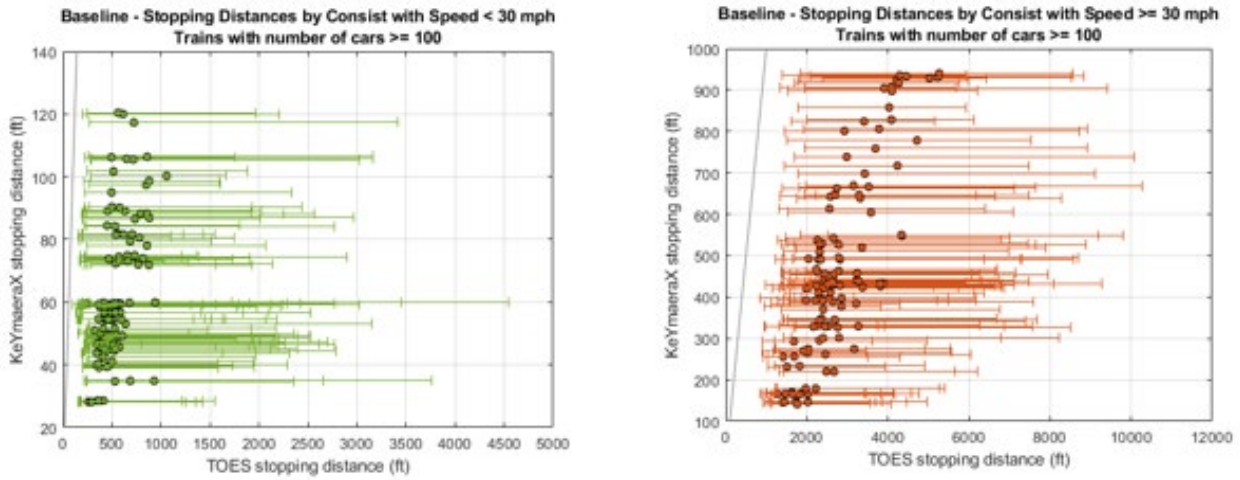


Figure 10. Baseline model instance – stopping distances of long trains with more than 100 cars.

V2: Delaybrake Model The delayed brake model under validation has two main improvements included. First, the train is not only determined as a single point mass at a certain location, but rather its weight is distributed over the train length. This concludes that the train accelerates on a slope depending on the average percent of grade under the train and the percentage of train weight on the average slope. Additionally, the total brake force is delayed considering the build-up time the train needs to reach the maximum brake force available. The build-up time in return is also calculated depending on the length of the train. Considering these improvements, the expectation of the validation is that the overall stopping distance would be more conservative, i.e., the train would start stopping earlier and therefore would need a longer stopping distance. More precisely, the stopping distance of longer trains is expected to considerably exceed the stopping distances of the baseline model. Figure 11 show the stopping distance of the delaybrake model filtered by trains with more than 100 cars.

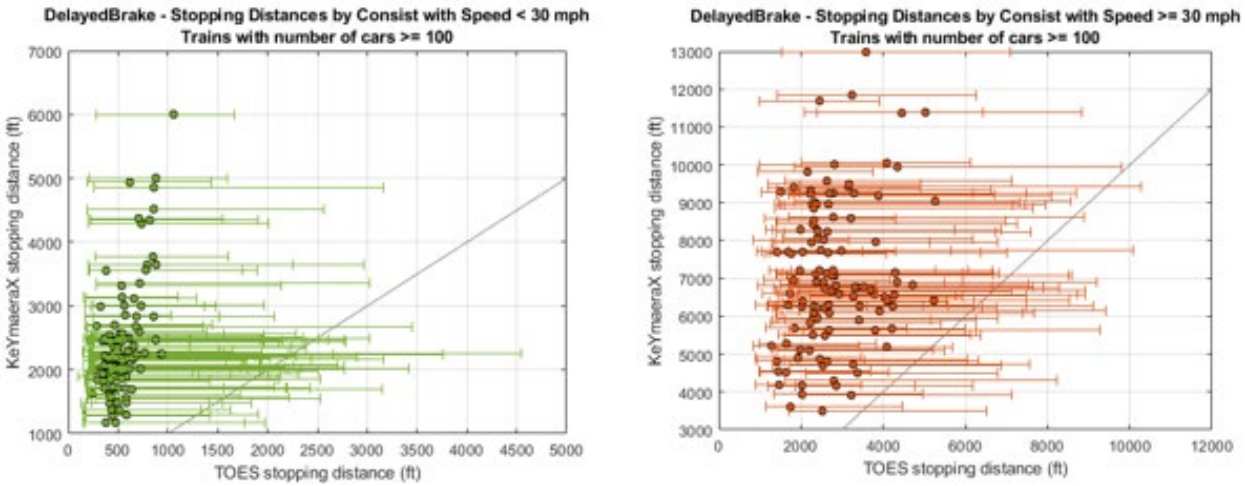


Figure 11. Delaybrake model instance – stopping distances of long trains with more than 100 cars.

Figure 12 summarizes all results by consist, with horizontal error bars again representing the internal variation of the TOES simulation results.

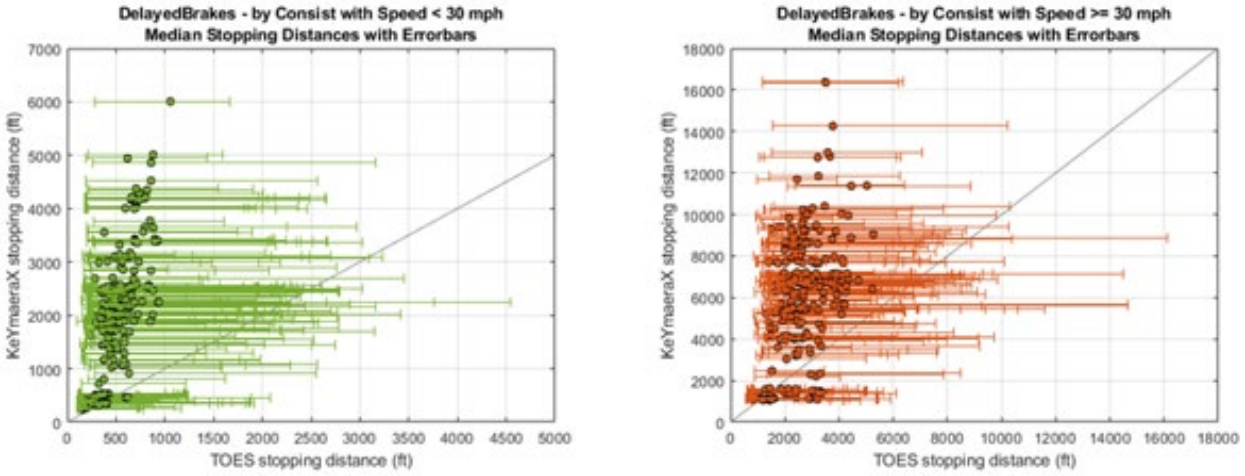


Figure 12. Delaybrake model stopping distances by consist (one circle per consist, horizontal position is median TOES distance, vertical is median model distance, error bars for silent variation).

As the results confirm, the delayed brake model has a significant shift of stopping distance to be more conservative. This also means that the model as parametrized results in safe stopping distances for most test scenarios. However, there are still many outliers with a large stopping distance which are safe but not practical as the train stops too far away from the stopping limit, as shown in Figure 13(b).

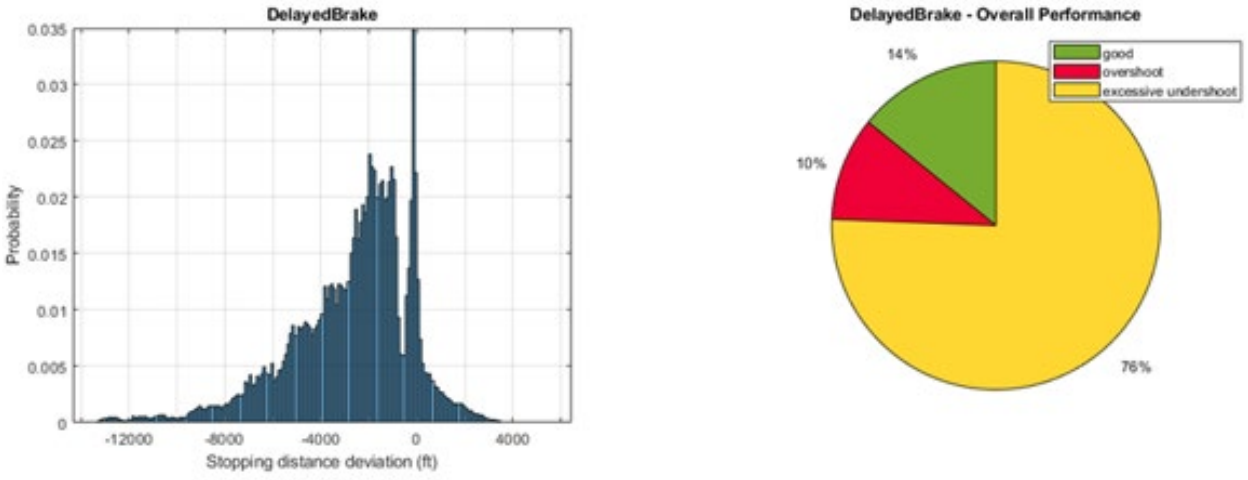


Figure 13. Delaybrake model summary.

Next, the stopping distance function Eq. (12) with air brake propagation is validated.

V3: Airbrake Model. The airbrake model includes the calculation of the air brake force build-up instead of a delayed brake application. The build-up time is a ramp-up function of a linear constant increasing brakeforce until it reaches a plateau, i.e., the maximal force to brake the train. Furthermore, the configuration of the model was adapted to reflect the braking power distribution with locomotives at different positions within the train. Thus, a train with locomotives in front and at the end of the train would also have two brake pipe valves which can drop the air pipe pressure from two different positions. In that case the air could flow through the valves of the front and end locomotive. Releasing air at two endpoints would also lower the time that required to reach the air pressure drop (typically 26psi) necessary for a full brake application.

Figure 14 compares the median stopping distances of the model instances to the TOES-simulated stopping distance, with horizontal error bars again representing the silent variation in the simulator.

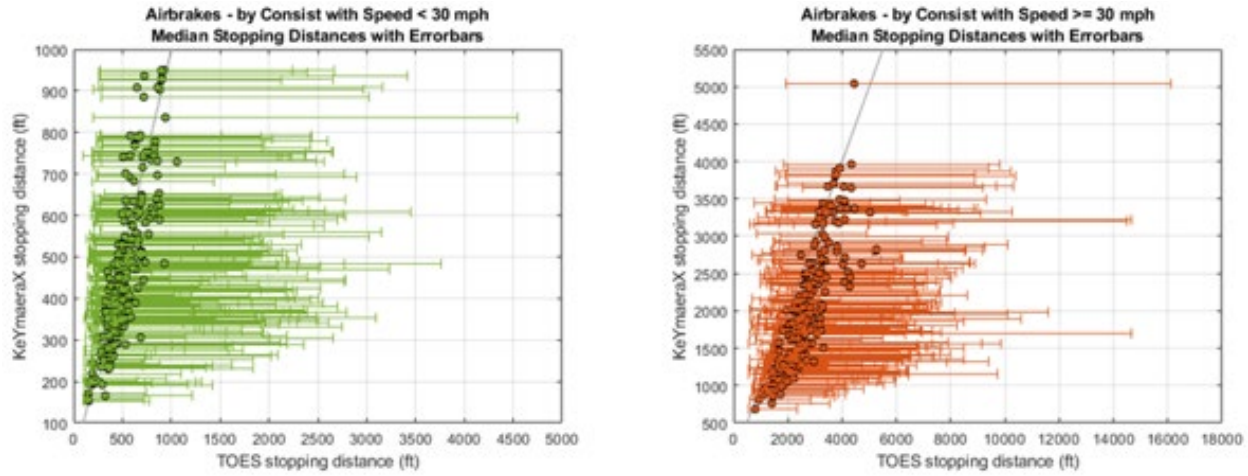


Figure 14. Airbrake model stopping distances by consist (one circle per consist, horizontal position is median TOES distance, vertical is median model distance, error bars for silent variation).

Figure 14 shows that the medians of the model instances align well with the simulated distance, indicating that the chosen parametrization closely fits the typical braking behavior of each consist. In this investigation the team also considered whether accounting for the position of the airbrake valves in parameterizing formal models better aligns stopping distances; the airbrake valves could be in the front (full train length when estimating brake application time); front and end (half train length when estimating brake application time); or front, middle, and end (1/3 train length when estimating brake application time). The results for head locomotives are shown in Figure 15, for head-end locomotives in Figure 16, and for head-middle-end locomotives in Figure 17.

As a result of the silent variation in the simulator, the summary in Figure 18 still classifies many simulation runs as overshoots, but the results in Figure 14, Figure 15, Figure 16, and Figure 17 show that the parametrization aligns the median stopping distances over each train's power distributions well with the median TOES stopping distances.

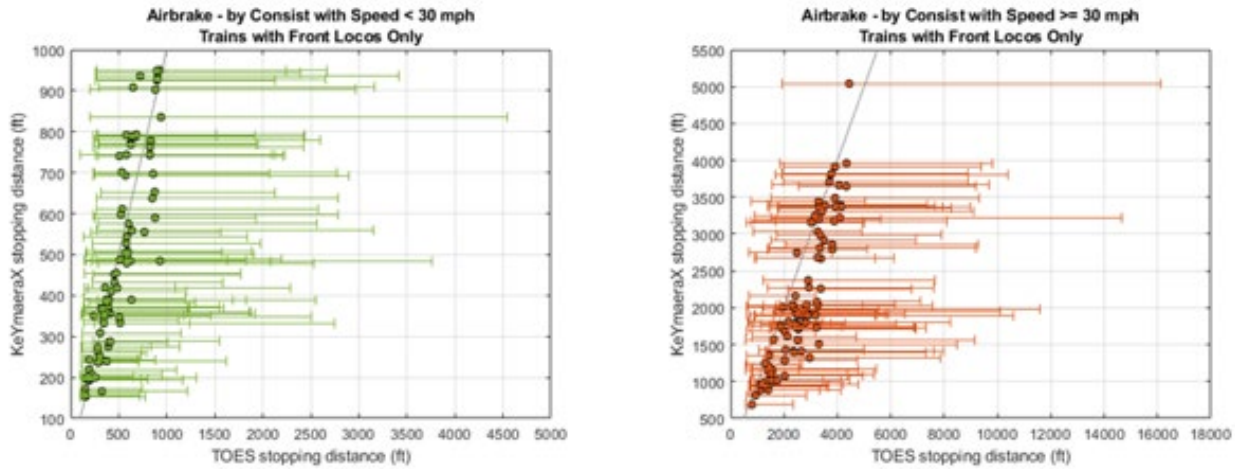


Figure 15. Airbrake model head locomotives – stopping distances by consist.

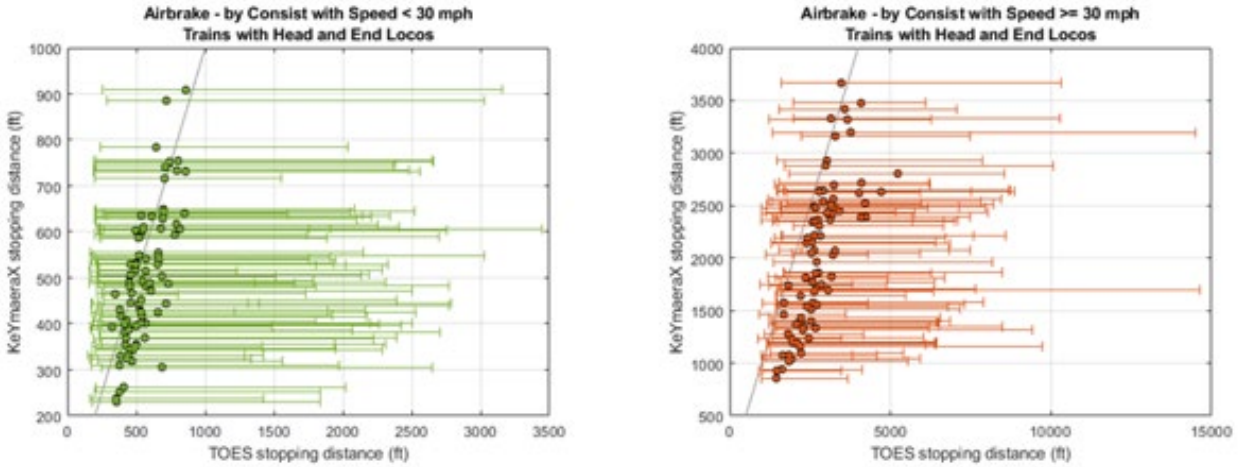


Figure 16. Airbrake model head-end locomotives – stopping distances by consist.

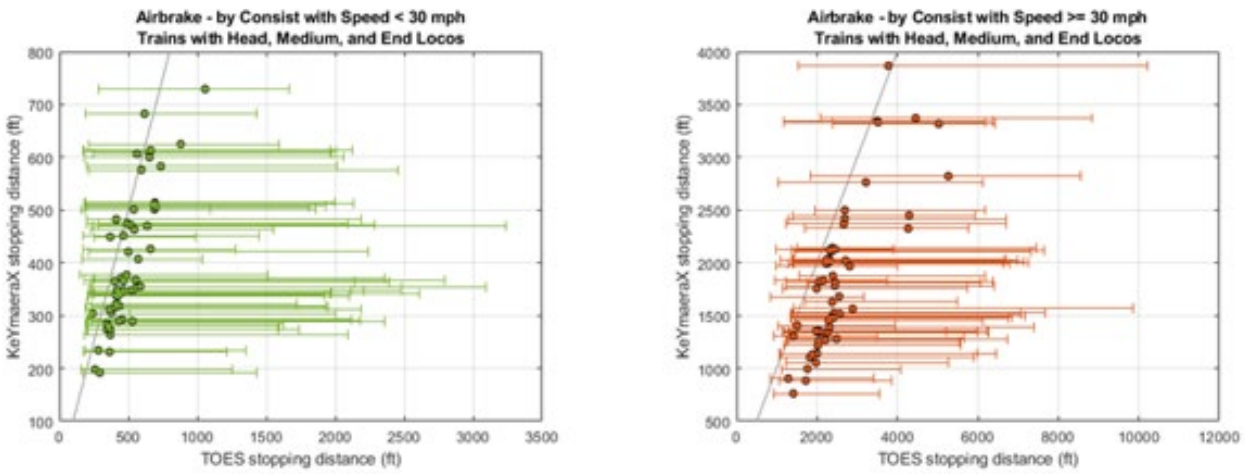


Figure 17. Airbrake model head-middle-end locomotives – stopping distances by consist.

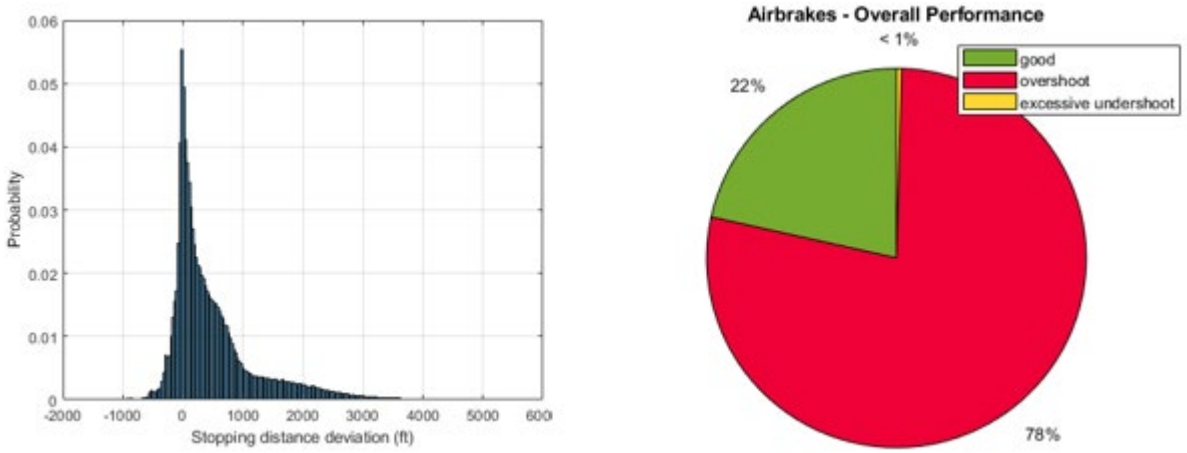


Figure 18. Airbrake model summary.

5.3 Meta-Analysis of the Testing-Based Parametrization Approach

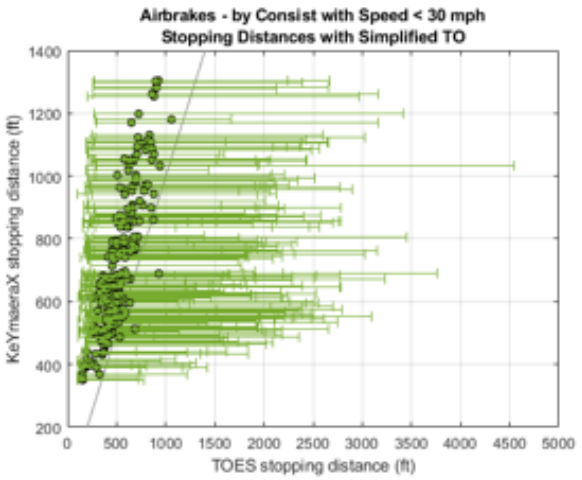
Parameter and Configuration Adjustments One of the identified challenges in validation is silent parameter variation in the simulator from identical starting configurations, which manifests itself as identical stopping distance predictions within a batch and is present as horizontal "lines" in the validation plots. The following parameters of the simulator configuration were used to create model instances:

- InitTrainVelocityMph,
- TrailingTons,
- Axles,
- NominalGrade,
- TrainLength,
- BrakeForceProvided,
- and LocoTonnage.

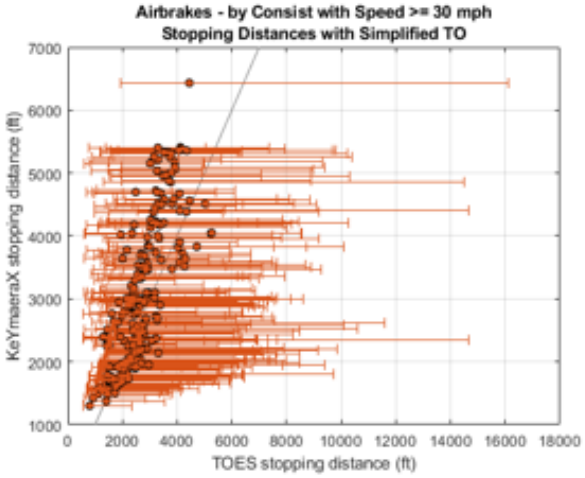
Figure 14 seems to indicate that higher velocity results in larger deviation between the model instance stopping distance and the simulator distance, especially some runs ≥30mph having little variation in the model instance stopping distance but large variation in the simulator distance (error bars in Figure 14). Close examination of the simulator configuration parameters that determine the model instance stopping prediction (PenaltyApplicationSpeedMph, TrailingTons, Axles, NominalGrade, TrainLength, BrakeForceProvided, and LocoTonnage) reveals that only BrakeForceProvided and PenaltyApplicationSpeedMph (±1mph) change slightly within a simulator batch. In 0.93 percent of runs, all parameters are identical and, consequently, the stopping distance predictions are all identical. All other runs have slight variation in the parameters, but considerably less than the effect of the silent internal variation in the simulator. Most prominently, BrakeForceProvided is used as the brake force that is available to the train when creating model instances, while the simulator may silently reduce this value considerably, see (Brosseau et al., 2013, Appendix D).

Parameter Uncertainties. The simulation runs for validation are based on the known parameters shared by the TOES simulation results. However, most of the train configuration parameters are unknown to the model instance because the simulator does not expose them. Internally, the TOES simulation uses a Monte Carlo Variation which specifically applies to cars regarding their weight, brakeforce, brake valve, etc. These unknown parameters influence how the starting configurations, such as BrakeForceProvided, are silently varied in the simulator, while the model instances use the simulator starting configurations as ground truth. As a result, the comparison between simulator and model instance can confirm trends, but there is no uniform way of choosing parameters based on the surrogate parameters that are visible outside the simulator. Instead, for true parametrization a better insight is needed from either a domain expert or other parametrization techniques.

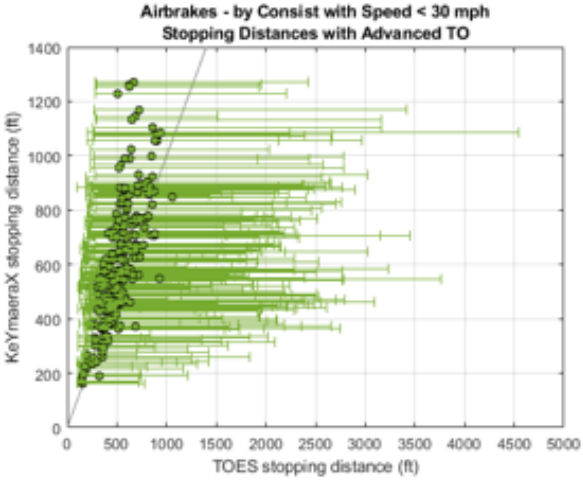
One way to compensate for these uncertainties is to add a margin to the stopping distances calculated to ensure that the train is within the identified safety requirements. Such a margin is defined by a target offset function. The amount of offset required to meet the requirements depends on the characteristics of the simulation scenarios. Previous research suggests a base (Brosseau et al., 2013. Eq. 4) and advanced offset function calculation (Brosseau et al., 2013. Eq. 8). Both have been applied to the formal model to see its theoretical possibility to meet the requirements. Figure 19 shows the target offset functions to the stopping distance predictions of model instances. The variability within the silent parameters is, however, still not fully accounted for by either of the two target offset functions.



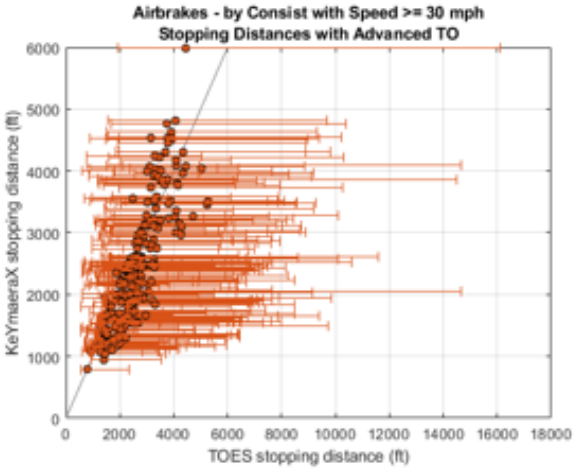
(a) Target offset Eq. (4) of (Brosseau, et al., 2013), speed < 30mph



(b) Target offset Eq. (4) of (Brosseau, et al., 2013), speed > 30mph



(c) Target offset Eq. (8) of (Brosseau, et al., 2013), speed < 30mph



(d) Target offset Eq. (8) of (Brosseau, et al., 2013), speed > 30mph

Figure 19. Air brakes with additional target offset according to (Brosseau J. , Ede, Pate, Wiley, & Drapa, 2013).

6. Simulation Architecture for Reinforcement Learning

The team developed a simulation environment for reinforcement learning to obtain agents that automatically engage brakes or agents to automatically issue movement authorities. This allows the train to stop without violating a desired stopping point (i.e., EOA), while at the same time optimizing for some configurable metric of interest (e.g. overall trip time, dwell time, etc.). The learned controller must ensure that the train is safe and does not overshoot the EOA. The development of such a reinforcement learning agent involved several major building blocks, including a reinforcement learning loop able to determine the current state of the agent from a set of pre-defined actions, as well as a train and track simulation environment able to interface with the reinforcement learning loop. This interface provides feedback and state updates based on the chosen action (e.g. accelerate, cruise, brake) while considering the previous state (e.g. position, speed, etc.) and the operating environment (e.g. train consist). The runtime performance of the simulation environment is crucial to the success of reinforcement learning. In this section, the technical architecture of the simulation environment for learning agents that automatically engage brakes is discussed. Later sections will explore the simulation setup for learning to issue movement authorities.

6.1 Simulation Environment Requirements

The reinforcement learning simulation environment requires an architecture that 1) accepts information about the train, track, and controller, 2) simulates train motion according to the decisions of the controller, and 3) outputs the train motion as state snapshots (e.g., position, speed, etc.). The minimum set of requirements for the initial version of the simulator environment are discussed below.

General Simulator Features. The simulator interface should be programmatic in that a software script interacts with the simulator directly, entering settings and performing runs without human intervention. For basic baseline model testing the simulator should satisfy the requirements below:

- **Programmatic parameter setting.** In this context, programmatic means that the simulator accepts a configuration file, or a script would directly interface with the simulator. A human would not need to manually enter settings.
- **Programmatic commands.** A command script pre-defines a sequence of control decisions at simulation start and the simulator automatically executes the commands in the correct order.
- **Interaction with controller.** The simulator accepts control decisions from an external controller and sends the simulation output (e.g., position, velocity etc.) to the external controller.
- **High fidelity.** Simulators with higher fidelity modeling are useful (for example, a simulator that permits car load specification would still work with unit trains, but a simulator that lets a train stop in an instant would not have enough fidelity). A simulator should allow setting varying track gradients and ideally also curvature profiles.

- **Configure control latency.** The frequency that the controller is allowed to programmatically decide between control choices (e.g., acceleration or braking) should be configurable. Preferably the latency would be at most a few seconds of model time.

Simulator Modes. The minimum requirement for learning to automatically stop is simulation of brake curves:

- **Brake curve one-shot simulation setup**: From parameter configuration, initial velocity, initial position, and brake force/deceleration, the simulator creates position, speed, brake force/deceleration updates/log until a full stop of the train, until a configured end position is reached, or until a configured number of simulation steps is performed.

6.2 Brake Curve Simulation Environment Architecture

Figure 20 shows the high-level architecture of the simulation environment which was developed to derive policies for a braking model.

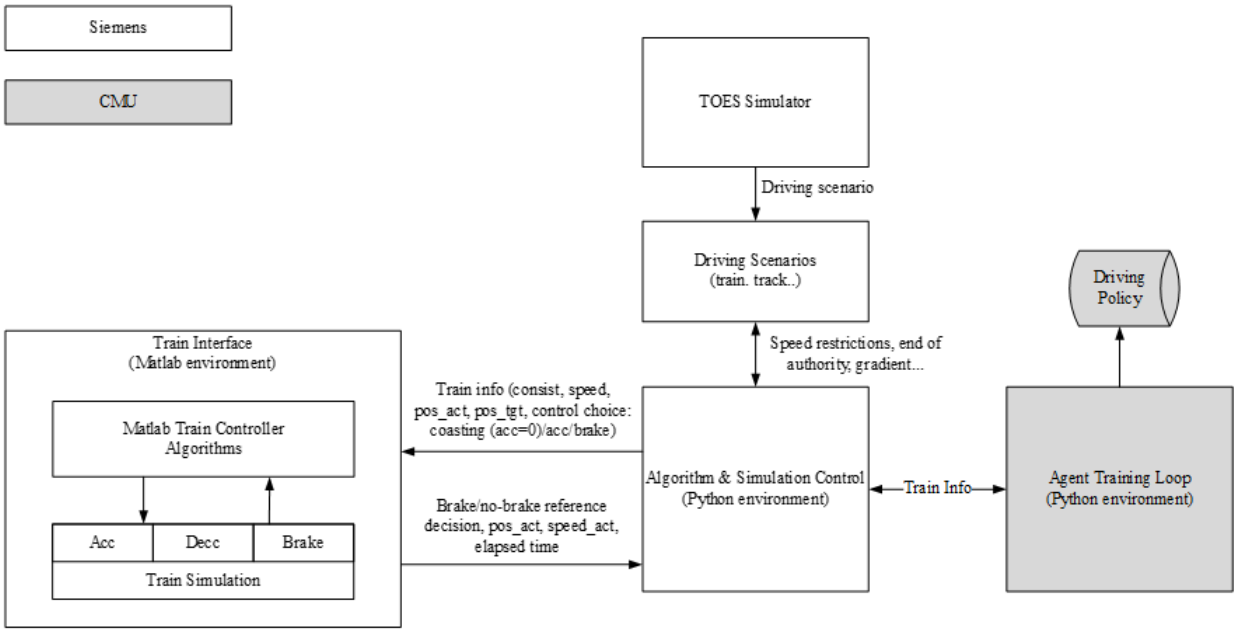


Figure 20. High-level architecture of the simulation environment.

The main components of the simulation setup are:

- **Agent Training Loop.** The Agent Training Loop learns to brake without overshooting the EOA, but also to minimize undershoot. It provides state up- dates, in particular current train position and the next command (e.g., acceleration / braking / coasting). During its run time, it calculates the reward/penalty based on feedback from the Train Interface and updates the driving policy (i.e., learning state).
- **Driving Scenario.** The driving scenario describes the train configuration (i.e., train consist), as well as the operating environment, in particular track information. The driving scenarios were derived from the TOES simulation runs used for the braking curve model validation.

- **Algorithm and Simulation Control.** The Algorithm and Simulation Control component is responsible for parsing the Driving Scenario from an external source (e.g., CSV file). It also provides API access to Agent Training Loop, parametrizes the MATLAB environment via the Train Interface, forwards braking distance / position calculations from Train Interface back to Agent Training Loop, and forwards command updates from Agent Training Loop to Train Interface.
- **Train Interface.** The Train Interface controls the MATLAB Train Controller Algorithm, calculates position updates based on last communicated or updated driving instruction from Algorithm and Simulation Control. Provides position updates back to Algorithm and Simulation Control.
- **MATLAB Train Controller Algorithm.** The Train Controller Algorithm calculates braking distance and acceleration. It also determines whether safe acceleration/coasting is possible or if braking should be initiated to maintain safety.

The Figure 21 shows the simulation control sequence between the Brake Curve Simulation (realized from the architecture in Figure 20) and the Agent Training Loop required for the three simulation steps. The Agent Training Loop provides commands (e.g., coast, coast, brake) with other parameters. In each step, the simulation provides feedback on the stopping distance, train position, speed etc., which is used by the agent for updating the driving policy. This is an example of how training is facilitated by the simulation.

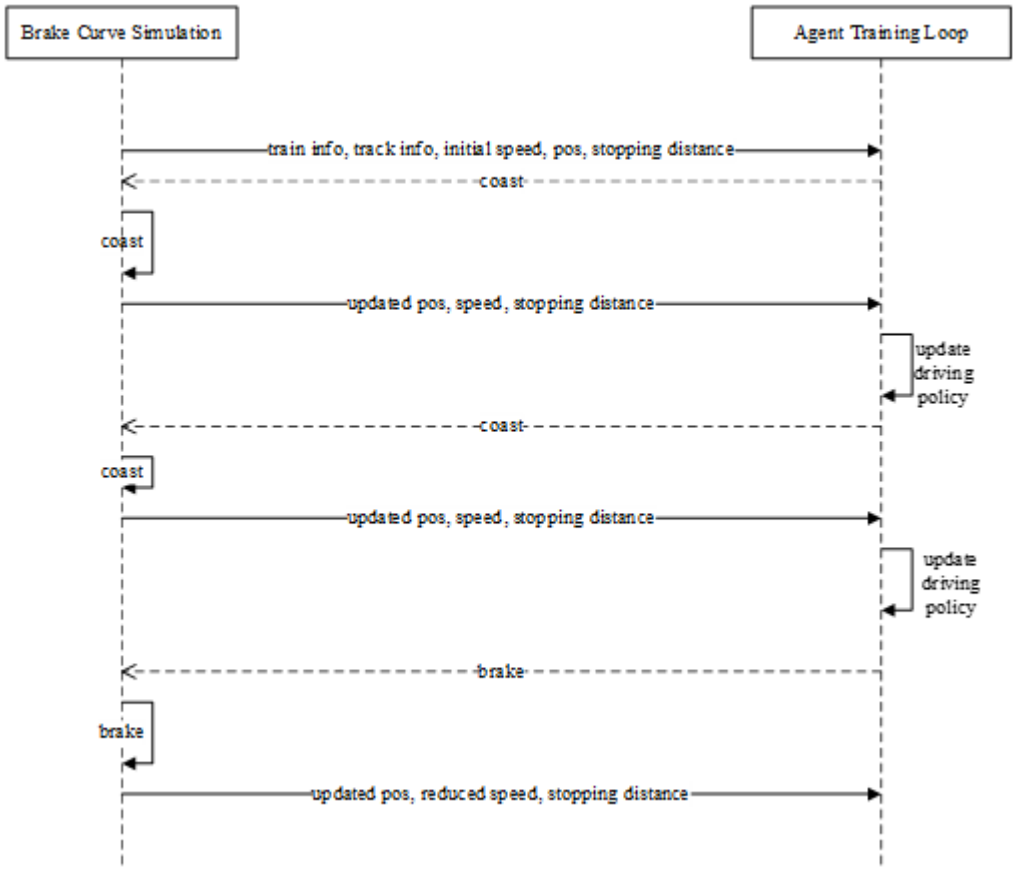


Figure 21. Sequence diagram illustrating a simulation control sequence for a training loop.

Architectural Decisions. A few important architectural decisions in the context of the above architecture are highlighted below:

- **Use of Python as a wrapper around a Matlab Controller Algorithm**. For performance reasons, the actual controller algorithms (e.g., braking curve calculation, acceleration, etc.) will be implemented in MATLAB. For ease of use and convenience during the integration with the Agent Training Loop, the MATLAB implementation provides an external Python interface to allow parametrization and control.
- **Export of MATLAB Train Controller as Python package rather than directly calling MATLAB functions from Python**. After some performance tests, the team determined that the MathWorks recommended interface between Python and MATLAB (using MATLAB Engine API) did not meet the performance requirements due to the time necessary to initialize the MATLAB interpreter. The team ran performance tests with 100 random data samples from Driving Scenarios to call the MATLAB Train Controller Algorithm from Algorithm and Simulation Control (see Figure 20) via the MATLAB Engine API. The average runtime was measured to be 0.1060 seconds. To mitigate this, the team exported the MATLAB algorithms and used them as Python packages. The average runtime after this was 0.0425 seconds, which was almost twice as fast than using MATLAB Engine to interface between the MATLAB Controller Algorithms and the Algorithm and Simulation Control Python environment.

Deployment Environment. To ensure uninterrupted access and the ability to run long-lasting training cycles, the simulation environment was deployed to an AWS EC2 instance, accessible via secure shell protocol (see Figure 22).

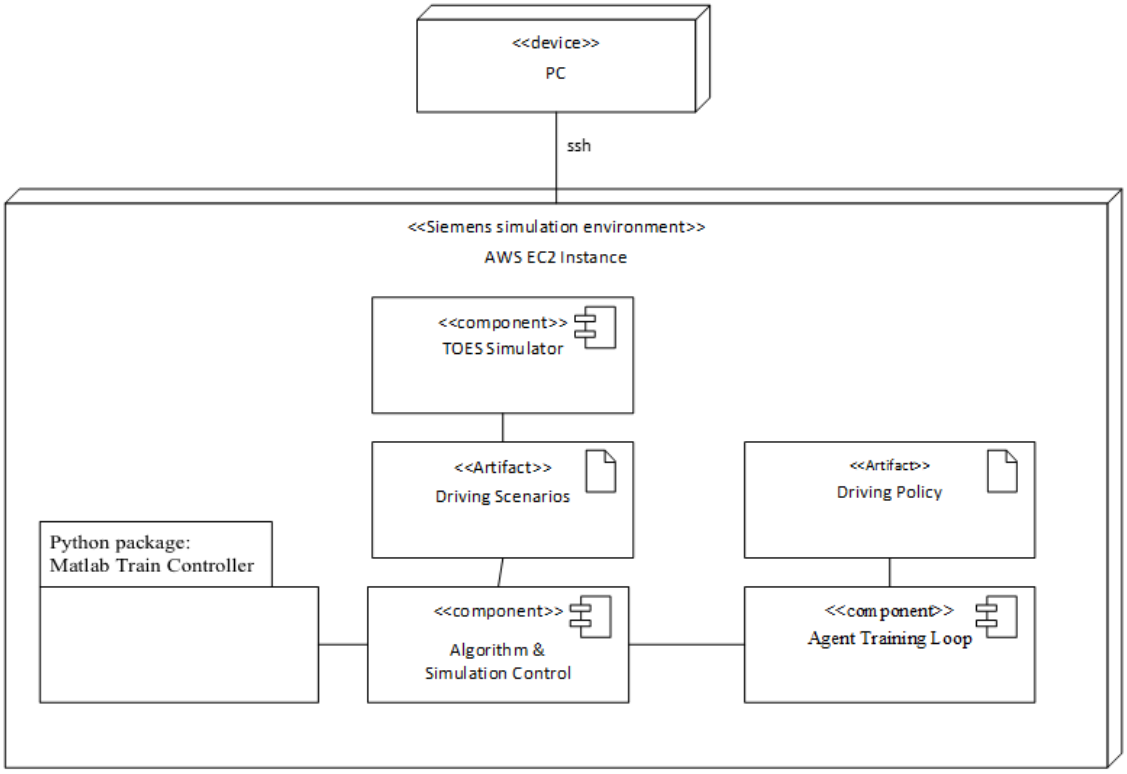


Figure 22. Deployment view of the brake curve simulation environment.

The proposed simulation architecture serves as a blueprint to the training of reinforcement learning agents. The setup has a proximity to a production environment. Two of the main requirements for the simulator were 1) performance, and 2) ease of use with the training agent. To address these, MATLAB controller algorithms were exported as a Python package and a python wrapper was provided. All the components were deployed and hosted in a cloud environment that provided an uninterrupted training environment.

The braking environment is a useful learning environment for emergency braking controllers that have a single action executed until standstill and cannot be interrupted or reverted. In the next section, the braking environment is extended with a choice of speeding up, which allows learning controllers to make multiple decisions.

6.3 Simplified Braking and Acceleration Environment

To train controllers to make decisions, follow them for a while, and then change their decision when external circumstances change, the team used a learning environment from the literature (Dohmen et al., 2021). This environment supports a continuous change of speed limits rather than a one-shot application of the air brakes. In comparison to Section 8.2, the simulator uses a simplified brake and acceleration model. The environment resembles a fixed-block setup, where speed limits are posted at fixed positions along the route. A full moving-block simulation setup in presented in Section 11.

Environment. LongiControl (Dohmen et al., 2021) provides a longitudinal vehicle control environment with state space $[\boldsymbol{x(t)}, \boldsymbol{v(t)}, \boldsymbol{a(t)}, \boldsymbol{a_{prev}(t)}, \boldsymbol{v_{lim}(t)}, \vec{\boldsymbol{v}}_{\boldsymbol{lim,fut}}\boldsymbol{(t)}, \vec{\boldsymbol{d}}_{\boldsymbol{lim,fut}}\boldsymbol{(t)}]$ of vehicle position $\boldsymbol{x(t)}$, speed $\boldsymbol{v(t)}$, acceleration $\boldsymbol{a(t)}$, previous acceleration $\boldsymbol{a_{prev}(t)}$, current speed limit $\boldsymbol{v_{lim}(t)}$, upcoming two speed limits $\vec{\boldsymbol{v}}_{\boldsymbol{lim,fut}}\boldsymbol{(t)}$, and distances to the upcoming two speed limits $\vec{\boldsymbol{d}}_{\boldsymbol{lim,fut}}\boldsymbol{(t)}$. The action space is continuous acceleration in the interval $\boldsymbol{[3, -3]m/s^2}$, and the sampling time is 0.1 s. The environment has posted speed limits of 50 km/h at [0, 250) m, 80 km/h at [250, 500) m, 40 km/h at [500, 750) m, and 50 km/h after 750 m.

The environment is used in Section 9 to evaluate an approach for turning the safety envelopes of formal models into safety components of a reward function for reinforcement learning.

7. Control Reinforcement Learning from Formal Models

The research team studied options of combining the verified safety envelope of the formal model with reinforcement learning. For such a combination, the verified safety envelope is viewed in a quantitative way in that it not only identifies whether the choices of a learned controller are safe or unsafe according to the formal model, but also measures the robustness of the choice (i.e., how safe or unsafe). This offers several benefits:

- Knowing whether the system is close to the safety boundary may help in understanding how aggressively control decisions can be made or how much unanticipated disturbance (e.g., friction, weight uncertainty, limited brake failure, etc.) can be tolerated. Close to the safety boundary there is little room for mistakes, so conservative control decisions are more likely to maintain safety. Well inside the safety envelope there is more room for tolerating unanticipated disturbances.
- Change in robustness may indicate the direction in which the system is evolving. Controllers and learning agents might be able to learn from how their decisions influence safety over time. In the safe case, controllers might be able to optimize for driving close to the safety boundary, making operations more efficient. When unsafe, controllers might be able to learn which decisions move operations back to safety and which decisions exacerbate the problem.
- Change in robustness under maintained decisions may indicate environment change and give early warning that control choices must be adapted when the true operation context begins deviating too far from the development, training, and testing environment.

Hybrid systems models of train control and motion were introduced in the previous sections. Those models are suitable to faithfully represent the continuous motion of trains and discrete control decisions in a uniform way and enable formal verification. There are, however, several challenges in creating robustness measures from a formal hybrid systems model. First, hybrid systems models include many different quantities (e.g., speed vs. position vs. weight) and it is not immediately obvious how to combine them into a single, meaningful robustness measure going beyond the formal model's purpose of classifying between safe and unsafe. Second, hybrid systems models have discontinuous jumps, which is a very useful feature to model switching from one operation mode to another (e.g., from driving to engaging air brakes), but causes discontinuity in the robustness measure. Since reinforcement learning works best with gradients so that the learning agent can experience change gradually (as opposed to signals that instantly change from safe to unsafe), such discontinuities make it challenging to use a quantitative robustness measure directly as part of the reward signal. Third, formal models do not distinguish between user preferences and hard safety constraints (e.g., some decisions are equally safe, but might differ in non-safety related features).

The team developed and studied approaches to signal rescaling to address user preferences and combining robustness measures of quantities with different units (e.g., position vs. speed vs. acceleration). The experiments indicate that signal rescaling is an important tool that can considerably influence learning outcomes.

A particularly attractive approach for training controllers is reinforcement learning, for its seemingly straightforward way of specifying desired behavior with a reward function. Designing

reward functions, however, is challenging, not least because they need to balance safety-oriented and goal-oriented requirements.

This section is based on (Qian & Mitsch, 2023). Ideas are used from logically constrained reinforcement learning (Hahn et al., 2020) (Hammond et al., 2021) (Hasanbeig et al., 2020) to develop a formal approach for generating the safety-oriented aspects of reward functions from the formal models of earlier sections (or any other verified predictive hybrid systems model). The main reasoning behind this approach is that hybrid systems models describe safety envelopes that can be turned into formally verified runtime monitors (Mitsch & Platzer, 2014). These runtime monitors not only distinguish safe from unsafe behavior, but with an appropriate quantitative interpretation can be used to measure the robustness of actions around these safety envelopes. The challenge in deriving a useful (for reinforcement learning) robustness measure from a formal model is that relative importance of safety aspects is not immediately obvious from the formal model alone, and differences in units makes comparison of the magnitude of robustness values across different aspects of the formal model difficult. For example, an autonomous vehicle model may encode brake force limits and speed limits; as the vehicle approaches a speed limit, it is acceptable to experience decreased robustness in brake limit to not violate the posted speed limit. Another challenge is that measure-zero safety aspects can hide progress or regression in other requirements.

The team addressed these challenges by adapting robustness measures from metric temporal logic (Fainekos & Pappas, 2006) and signal temporal logic Donze et al., 2013), and by developing signal rescaling (Tran et al., 2019) operators to adjust the relative importance of competing safety aspects. The benefits of this approach are that the safety specification is rigorously checked for correctness and the resulting reward function inherits the predictive nature of the hybrid systems model and its safety guarantees. The approach is based on three actions: 1) based on (Mitsch & Platzer, 2014) (Fainekos & Pappas, 2006), a quantitative interpretation of hybrid systems models with an account for measure-zero requirements is developed; 2) signal rescaling (Tran et al., 2019) operators to specify the relative importance of (competing) safety aspects in the formal model are developed; and 3) the approach on a standard reinforcement learning environment for vehicle control (Dohmen et al., 2021) is evaluated.

A complementary approach is taken to shielding by interpreting hybrid systems models quantitatively through their relational abstraction as ModelPlex monitors. To this end, a hybrid systems normal form is defined that is designed to align states of the hybrid systems model with states in the training process.

Definition 1 (Time-triggered normal form). *A hybrid systems model α in time-triggered normal form is of the shape $(u := ctrl(x); t := 0; \{x' = f(x, u), t' = 1 \,\&\, t \leq T\})*$, where $u := ctrl(x)$ is a discrete hybrid systems model not mentioning differential equations.*

A hybrid program in time-triggered normal form models repeated interaction between a discrete controller $u := ctrl(x)$ that acts with a latency of at most time T and a continuous model $t := 0; \{x' = f(x, u), t' = 1 \,\&\, t \leq T\}$ that responds to the control choice u. Note that the discrete controller $u := ctrl(x)$ typically focuses on safety-relevant features (e.g., collision avoidance) while abstracting from goal-oriented features (e.g., desired cruise speed). The goal was to develop a principled approach to reward shaping to translate the safety-relevant insights of formal verification to reinforcement learning. Figure 23 illustrates how the states of a formal model correspond to states in reinforcement learning (note that unlike in usual RL notation,

where states are responses of the environment and actions are drawn from a separate set, the states of the formal model include the values of actions).

Let $\boldsymbol{s_0} \in \boldsymbol{S}$ be the state before executing $\boldsymbol{u := ctrl(x)}$, $\boldsymbol{s_1} \in \boldsymbol{S}$ be the state after executing $\boldsymbol{u := ctrl(x)}$ (and before executing $t := 0; \{x' = f(x, u), t' = 1 \,\&\, t \leq T\}$), and $\boldsymbol{s_2} \in \boldsymbol{S}$ be the state after executing $t := 0; \{x' = f(x, u), t' = 1 \,\&\, t \leq T\}$ (which becomes $\boldsymbol{s_0}$ in the next loop iteration).

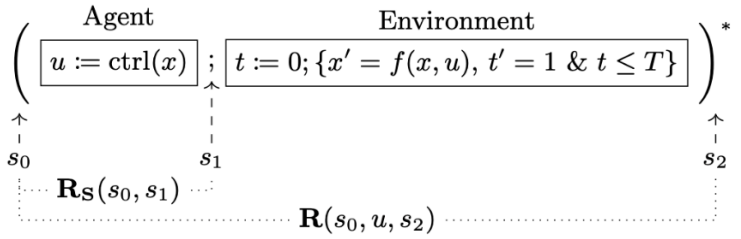


Figure 23. Overview of aligning states in the formal model and the training process. The reward $\boldsymbol{R(s_0, u, s_2)}$ is given for the transition $\boldsymbol{(s_0, u, s_2)}$, whereas the states of a formal model include the values of actions, so action $\boldsymbol{u}$ traverses to intermediate state $\boldsymbol{s_1}$ from which the environment responds by producing state $\boldsymbol{s_2}$. Therefore, we can give separate reward $\boldsymbol{R_S(s_0, s_1)}$ from the predictive formal model for choosing action $\boldsymbol{u}$ in state $\boldsymbol{s_0}$.

These states relate to the training process in reinforcement learning as illustrated in Figure 5: $\boldsymbol{s_0}$ corresponds to the state before the agent picks an action, $\boldsymbol{s_1}$ is the state after the agent chose an action $\boldsymbol{u} \in \boldsymbol{A}$ (but before it is actuated in the environment), and $\boldsymbol{s_2}$ is the result state of executing the action in the environment. In typical reinforcement learning setups, the reward associated with the transition $\boldsymbol{(s_0, u, s_2)}$ is computed by a reward function $\boldsymbol{R}: \boldsymbol{S \times A \times S \to \mathbb{R}}$. We provide separate reward $\boldsymbol{R_S}: \boldsymbol{S \times S \to \mathbb{R}}$ directly for choosing action $\boldsymbol{u}$ in state $\boldsymbol{s_0}$ from the predictive formal hybrid systems model, which requires a quantitative interpretation of ModelPlex formulas, as discussed below.

7.1 Quantitative Interpretation of ModelPlex Formulas

MTL/STL robustness measures (Fainekos & Pappas, 2006) (Fainekos & Pappas, 2009) (Donze et al., 2013) are adapted to define a quantitative interpretation of ModelPlex formulas, which describes how robustly satisfied a monitor is over two states.

Definition 2 (Quantitative ModelPlex). The function $Q: M \to \mathrm{T}$ interprets a ModelPlex formula $\varphi \in M$ quantitatively as an arithmetic expression $\theta \in \mathrm{T}$ in $+, -, \cdot, /$ over the reals ($\theta: S \times S \to \mathbb{R}$):

$$\mathbf{Q}(\theta \geq 0)(s_0, s_1) \triangleright \theta(s_0, s_1)$$
$$\mathbf{Q}(\theta > 0)(s_0, s_1) \triangleright \theta(s_0, s_1)$$
$$\mathbf{Q}(\theta = 0)(s_0, s_1) \triangleright \mathbf{Q}(\theta \geq 0 \wedge -\theta \geq 0)(s_0, s_1)$$
$$\mathbf{Q}(\theta \neq 0)(s_0, s_1) \triangleright \mathbf{Q}(\theta > 0 \vee -\theta > 0)(s_0, s_1)$$

$$\mathbf{Q}(\phi \wedge \psi)(s_0, s_1) \triangleright \begin{cases} \mathbf{Q}(\psi)(s_0, s_1) & \text{if } \phi \equiv \theta{=}0 \text{ and } (s_0, s_1) \models \theta{=}0 \\ -|\theta(s_0, s_1)| & \text{if } \phi \equiv \theta{=}0 \text{ and } (s_0, s_1) \not\models \theta{=}0 \\ (\mathbf{Q}(\phi) \sqcap \mathbf{Q}(\psi))(s_0, s_1) & \text{otherwise} \end{cases}$$

$$\mathbf{Q}(\phi \vee \psi)(s_0, s_1) \triangleright (\mathbf{Q}(\phi) \sqcup \mathbf{Q}(\psi))(s_0, s_1)$$
$$\mathbf{Q}(\neg\phi)(s_0, s_1) \triangleright -\mathbf{Q}(\psi)(s_0, s_1)$$

where $\sqcup$ is max, $\sqcap$ is min, the atomic propositions in φ are normalized to $\theta = 0, \theta \neq 0, \theta \geq 0, \theta > 0$, and conjunctions are reordered to list all $\theta = 0$ before inequalities.

Note that equalities $\theta = 0$ result in measure-zero robustness when they are satisfied: in a sense, their robustness is only meaningful when violated, since there is only a single way to satisfy $\theta = 0$. In conjunctions of the shape $\theta = 0 \wedge \varphi$, the chosen robustness definition avoids unnecessary measure-zero robustness by evaluating to $\boldsymbol{Q}(\varphi)$ when $\theta = 0$ is satisfied. In contrast, the naive phrasing $\boldsymbol{Q}(\theta = 0 \wedge \varphi) \rhd \min(\boldsymbol{Q}(\theta \geq 0), \boldsymbol{Q}(\varphi))$ would evaluate to 0 when $\theta = 0$ is satisfied and thus hide changes in robustness in φ, which can be valuable reward signals to the agent. Quantitative ModelPlex maintains safety by overapproximating the original monitor verdict, see Lemma 1.

Lemma 1 (Mixed Quantitative ModelPlex Overapproximates Verdict). The quantitative interpretation maintains the monitor verdict, i.e., the following formulas are valid: $\boldsymbol{Q}(\varphi) > 0 \rightarrow \varphi$ and $\boldsymbol{Q}(\varphi) < 0 \rightarrow \neg\varphi$ for all ModelPlex formulas $\varphi \in M$.

Proof. By structural induction on ModelPlex formula and term operators.

Mixed inequalities in Lemma 1 require for the quantitative interpretation to be conservative in the sense of causing false alarms on weak inequalities (a robustness measure of 0 is inconclusive). When comparisons are restricted to only weak or only strict inequalities, we maintain equivalence between the quantitative and the Boolean interpretation of ModelPlex (see Corollary 1).

Corollary 1 (Weak/Strict Quantitative ModelPlex Maintains Verdict). When comparisons are restricted to weak/strict inequalities, the quantitative interpretation maintains the monitor verdict, i.e., the following formula is valid for weak inequalities $\boldsymbol{Q}(\varphi) \geq 0 \leftrightarrow \varphi$ (for strict inequalities $\boldsymbol{Q}(\varphi) > 0 \leftrightarrow \varphi$ for all ModelPlex formulas $\varphi \in M$ restricted to weak inequalities $\geq, =$ in atomic propositions (strict inequalities $>, \neq$, respectively) and Boolean connectives $\wedge, \vee$.

Proof. By structural induction on ModelPlex formula and term operators.

With a quantitative interpretation of how robustly the choices of a reinforcement learning agent satisfy the formal model, several ways of combining the safety reward with other goal-oriented reward elements can now be discussed.

7.2 Logical Constraint Reward

To include feedback about the constraints of a formal model into the reward function, the agent receives goal-oriented reward when operating safely according to the model (safety monitor is satisfied) but receives the goal-oriented reward adjusted with a penalty from the monitor when operating unsafely.

Definition 3 (Logical Constraint Reward). Let $P \in M$ be a ModelPlex formula for $\langle u := ctrl(x)\rangle(\wedge(x \in BV(u := ctrl(x))x = x^+)$, let $\boldsymbol{R_G}$ be a goal-oriented reward function, and $\boldsymbol{R_S} = \boldsymbol{Q}(P)$ be the safety reward function from the quantitative interpretation of P. Let s_0, s_1, s_2 be the states before the agent chooses an action, after it chooses an action, and after the action is executed in the environment, respectively. The logical constraint reward is defined as:

$$\boldsymbol{R}(\boldsymbol{s_0}, \boldsymbol{s_1}, \boldsymbol{s_2}) = \begin{cases} \boldsymbol{R_G}(\boldsymbol{s_0}, \boldsymbol{s_2}) & \text{if } (\boldsymbol{s_0}, \boldsymbol{s_1}) \vDash \boldsymbol{P} \\ \boldsymbol{R_G}(\boldsymbol{s_0}, \boldsymbol{s_2}) + \boldsymbol{R_S}(\boldsymbol{s_0}, \boldsymbol{s_1}) & \text{otherwise} \end{cases}$$

Def. 3 discourages the agent to violate the assumptions of the formal model, while ignoring the safety reward when satisfied. The intuition behind this is that positive reward from the formal model is largest when robustly inside the boundary of the formal model's safety envelope, which may encourage the agent to make overly cautious (i.e., robust) action choices instead of making progress. Whether Def. 3 prioritizes goal-oriented reward or safety-oriented reward is entirely determined by the relative magnitude of $\boldsymbol{R_G(s_0, s_2)}$ vs. $\boldsymbol{R_S(s_0, s_1)}$. This can sometimes be undesirable since it does not necessarily encourage the agent to avoid safety violations. To emphasize safety and prioritize some aspects of the formal model over other aspects (e.g., satisfying a speed limit vs. satisfying deceleration assumptions) reward scaling is introduced to change the magnitude of rewards while preserving the safety verdict.

7.3 Reward Scaling

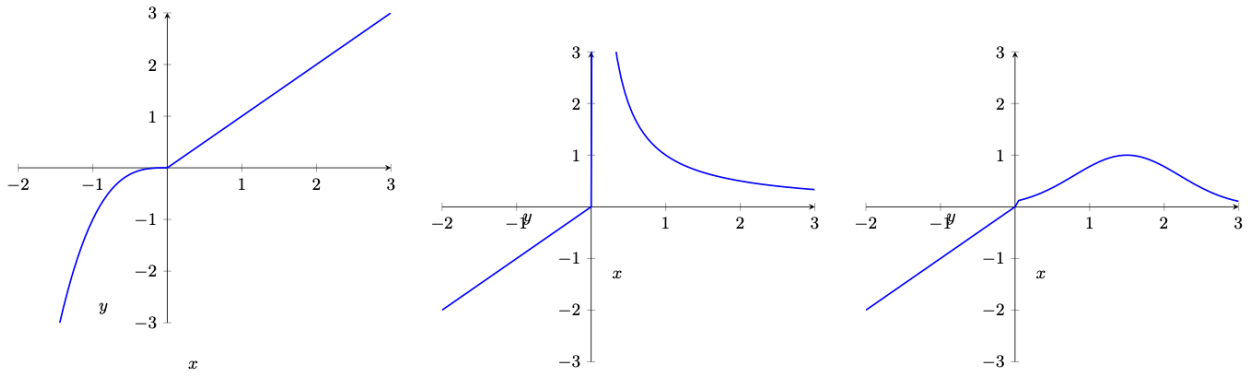
The logical constraint reward in Section 9.2 does not prioritize goal- vs. safety-oriented reward. Moreover, ModelPlex monitors do not distinguish between quantities of different units and sort in the formal model (e.g., a monitor conflates acceleration verdict, speed verdict, and position verdict into a single robustness measure). When used in a reward function, however, some safety aspects may be prioritized. For example, "violating" the brake assumptions of the formal model by having better brakes is acceptable and should not be penalized as seriously as violating a speed limit. Here, signal rescaling functions (Zhang et al., 2021) are developed to emphasize the verdict of the entire monitor or certain aspects of it.

Definition 4 (Scaling Function). A scaling function $\boldsymbol{C}: \mathbb{R} \to \mathbb{R}$ scales the result of a logical constraint reward function $\boldsymbol{R_S}$ such that the sign of its verdict remains unchanged, i.e., $\boldsymbol{C(0) = 0}$ and $\forall \boldsymbol{r \neq 0}. \boldsymbol{C(r)r > 0}$.

Note that the examples below use $\boldsymbol{C(\theta)}$ as a notation to indicate that the scaling function applies to a specific term θ.

Example 1 (Strong penalty). To emphasize that violating the formal model is highly undesirable, the following scaling function penalizes monitor violations while maintaining rewards for safety, as shown in Figure 24(a).

$$\boldsymbol{C\big(R_S(s_0, s_1)\big)} = \begin{cases} \boldsymbol{R_S(s_0, s_1)} & \text{if } \boldsymbol{R_S(s_0, s_1) > 0} \\ \boldsymbol{R_S(s_0, s_1)^3} & \text{otherwise} \end{cases}$$



(a) Strong penalty (b) Boundary preference (c) Distance preference

Figure 24. Reward scaling function illustrations.

Example 2 (Boundary preference). To encourage behavior that follows close to a safety boundary (i.e. drive close to but not past a speed limit), the following scaling function gives more reward when the state's boundary distance is very small while giving less reward otherwise (see Figure 24b):

$$\boldsymbol{C}\big(\boldsymbol{R_S}(\boldsymbol{s_0},\boldsymbol{s_1})\big) = \begin{cases} \dfrac{\mathbf{1}}{\boldsymbol{R_S}(\boldsymbol{s_0},\boldsymbol{s_1})} & \text{if } \boldsymbol{R_S}(\boldsymbol{s_0},\boldsymbol{s_1}) > \mathbf{0} \\ \boldsymbol{R_S}(\boldsymbol{s_0},\boldsymbol{s_1}) & \text{otherwise} \end{cases}$$

Example 3 (Distance preference). To encourage behavior that follows a certain distance from a safety boundary (i.e., drive 10 km/h below but not past a speed limit), the following scaling function gives more reward when the state's boundary distance is close to the desired distance while giving less reward otherwise (see Figure 24c):

$$\boldsymbol{C}\big(\boldsymbol{R_S}(\boldsymbol{s_0},\boldsymbol{s_1})\big) = \begin{cases} \boldsymbol{e}^{(-\boldsymbol{R_S}(\boldsymbol{s_0},\boldsymbol{s_1})-\mathbf{1.5})^2} & \text{if } \boldsymbol{R_S}(\boldsymbol{s_0},\boldsymbol{s_1}) > \mathbf{0} \\ \boldsymbol{R_S}(\boldsymbol{s_0},\boldsymbol{s_1}) & \text{otherwise} \end{cases}$$

Uniform scaling maintains the sign of the monitor verdict but emphasizes its importance relative to the goal-oriented components of a reward function. If the goal-oriented reward is designed to encourage reaching a goal fast (i.e., $\boldsymbol{R_G}(\boldsymbol{s_0},\boldsymbol{s_2}) \leq \mathbf{0}$ for all states $\boldsymbol{s_0},\boldsymbol{s_2} \in \boldsymbol{S}$), the logical constraint reward should be restricted to safety violations and offset to "exceed" the goal-oriented reward: $\mathbf{min}(\boldsymbol{R_S}(\boldsymbol{s_0},\boldsymbol{s_1}),\mathbf{0}) - |\boldsymbol{\delta}|$, where $\boldsymbol{\delta}$ is the minimum attainable goal-oriented reward.

The states $\boldsymbol{s}$ checked with a monitor $\boldsymbol{R_S}$ are composed of different aspects of the environment and agent behavior that can be scaled component-wise within $\boldsymbol{R_S}$ so certain aspects are given more weight in the monitor verdict.

If desired, reward scaling can decrease the importance of safety through decreasing the penalty for unsafe behavior, implying that violating safety for brief moments in time is allowed. This behavior appeared briefly in experiments when penalization for violating safety constraints was too low.

Definition 5 (Component-wise Scaling). Scaling $\boldsymbol{C_{\downarrow V}}\colon \mathbf{T} \to \mathbf{T}$ applies scaling function $\boldsymbol{C}$ to components in variables $\boldsymbol{V}$ of a safety-reward function $\boldsymbol{R_S}$:

$$C_{\downarrow V}(\mathbf{R_S}) = \begin{cases} \min(f,g) \triangleright \min(C_{\downarrow V}(f), C_{\downarrow V}(g)) & \\ \max(f,g) \triangleright \max(C_{\downarrow V}(f), C_{\downarrow V}(g)) & \\ f \triangleright C(f) & \textit{if } FV(f) \subseteq V \\ f \triangleright f & \textit{otherwise} \end{cases}$$

Component-wise scaling maintains the sign of safety-reward function $\boldsymbol{R_S}$:

$$\forall \boldsymbol{s_0},\boldsymbol{s_1} \in \boldsymbol{S}(\ \boldsymbol{R_S}(\boldsymbol{s_0},\boldsymbol{s_1}) = \mathbf{0} \wedge \boldsymbol{C_{\downarrow V}}(\boldsymbol{R_S})(\boldsymbol{s_0},\boldsymbol{s_1}) = \mathbf{0}$$
$$\vee\, \boldsymbol{R_S}(\boldsymbol{s_0},\boldsymbol{s_1}) \neq \mathbf{0} \wedge \boldsymbol{C_{\downarrow V}}(\boldsymbol{R_S})(\boldsymbol{s_0},\boldsymbol{s_1}) \cdot \boldsymbol{R_S}(\boldsymbol{s_0},\boldsymbol{s_1}) > \mathbf{0})$$

Example 4 (Emphasizing speed). To emphasize that violating a speed limit is unsafe, while being close to it is desirable, the difference between current speed $\boldsymbol{v}$ and speed limit $\boldsymbol{v_{des}}$ is scaled as follows:

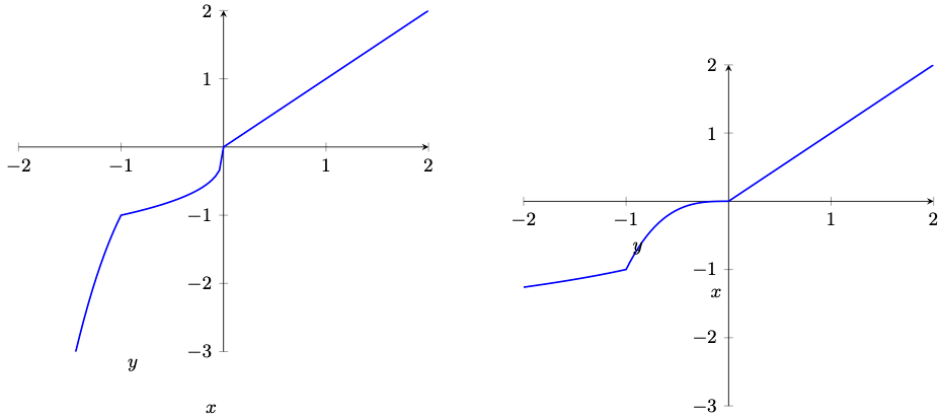
$$C_1(v_{\text{des}} - v) = \begin{cases} (v_{\text{des}} - v) & \text{if } v_{\text{des}} - v > 0 \\ (v_{\text{des}} - v)^{1/3} & \text{if } 0 \geq v_{\text{des}} - v > -1 \\ (v_{\text{des}} - v)^3 & \text{otherwise} \end{cases}$$

$$C_2(v - v_{\text{des}}) = \begin{cases} (v - v_{\text{des}}) & \text{if } v - v_{\text{des}} > 0 \\ (v - v_{\text{des}})^3 & \text{if } 0 \geq v - v_{\text{des}} > -1 \\ (v - v_{\text{des}})^{1/3} & \text{otherwise} \end{cases}$$

The above functions scale two different speed verdicts, $\boldsymbol{v - v_{des}}$ and $\boldsymbol{v_{des} - v}$, which in a formal model and thus a monitor may arise for different control choices (e.g., requiring to slow down when current velocity, $\boldsymbol{v}$, exceeds the speed limit, $\boldsymbol{v_{des}}$ vs. allowing to speed up when $\boldsymbol{v < v_{des}}$). When $\boldsymbol{v_{des} - v \leq 0}$, safety is violated and therefore $\boldsymbol{C_1}$ scales the negative verdict more aggressively by applying an exponential function. It is undesirable when $\boldsymbol{v - v_{des} \leq 0}$, safety is satisfied, but going much slower than v_{des}; therefore, $\boldsymbol{C_2}$ scales the negative verdict using a fractional exponential function (see Figure 25 below).

7.4 Potential-based Logical Constraint Reward

Reward shaping may cause unexpected behavior from the trained agent, as the reward directly influences what actions the agent takes and may cause the agent to learn a suboptimal policy (Ng et al., n.d.). To prevent unforeseen and unwanted behavior, potential-based reward shaping (Ng et al., n.d.) provides additional reward to the agent while guaranteeing the agent will learn the optimal policy of the original reward function. The additional reward is specified using the transition from the current to the future state, and this transition is formalized as the difference in "potential" of the two states (see Figure 25).



(a) $C_1(v_{\text{des}} - v)$ (b) $C_2(v - v_{\text{des}})$

Figure 25. Component-wise scaling to emphasize speed rewards.

Let $\boldsymbol{\Phi(s)}$ characterize certain features of a state $\boldsymbol{s \in S}$ (e.g., safety). Potential-based reward is then $\boldsymbol{R_P(s, a, s') = \gamma\Phi(s') - \Phi(s)}$, where $\boldsymbol{s}$ is the current state, $\boldsymbol{a}$ is the action taken to reach a future state $\boldsymbol{s'}$, and $\boldsymbol{\gamma}$ is some discount factor, which gets added to the original reward: $\boldsymbol{R(s, a, s') + R_P(s, a, s')}$.

The main reason (Ng et al., n.d.) why the policy is still optimal under this modified reward function is that the potential function itself does not prefer one policy over the other. Therefore, the original optimal policy is still preferred when the potential difference is added to the original reward function.

Example 5 (Potential-based Logical Constraint Reward). The original reward function $\boldsymbol{R(s, a, s')}$ is augmented with additional reward that is calculated using the predictive formal model. The loop iterations of the model in time-triggered normal form are aligned with the learning states $\boldsymbol{s}$ and $\boldsymbol{s'}$ of the reward function: let $\boldsymbol{s_0, s_1, s_2 = s}$ be the states from the formal model leading up to learning state $\boldsymbol{s}$, and let $\boldsymbol{s'_0 = s_2 = s, s'_1, s'_2 = s'}$ be the states of the formal model corresponding to the transition (s, a, s′). Since monitors are evaluated over two model states, the safety potential associated with learning state $\boldsymbol{s}$ is a function of the previous action $\boldsymbol{\Phi(s) = R_S(s_0, s_1)}$, whereas the safety potential associated with learning state $\boldsymbol{s'}$ is according to the safety of the current action $\boldsymbol{\Phi(s') = R_S(s'_0, s'_1)}$. Then, the additional reward is $\boldsymbol{R_P(s, a, s') = \gamma\Phi(s') - \Phi(s)}$ with $\boldsymbol{\gamma = 1}$, and the final reward function is: $\boldsymbol{R'(s, a, s') = R(s, a, s') + R_P(s, a, s')}$.

7.5 Evaluation

Formal model. In this evaluation, an existing formal model of a train protection system (Platzer & Quezel, 2009) is adapted and applied to a standard reinforcement learning environment for longitudinal vehicle control (Dohmen et al., 2021). The goal of the agent in longitudinal vehicle control is to drive forward as fast as possible while respecting posted speed limits. The speed limit control model (Model 1) comes with 3 components: a speed controller $\boldsymbol{spd}$ in lines 1–2 chooses acceleration to a desired cruising speed $\boldsymbol{v_{des}}$, the automatic train protection $\boldsymbol{atp}$ in line 3 may override the choice of $\boldsymbol{spd}$ if the remaining distance to the speed limit $\boldsymbol{d}$ is unsafe, and the motion model $\boldsymbol{drive}$ in lines 4–5 describes how the vehicle position $\boldsymbol{p}$ and speed $\boldsymbol{v}$ change in response to the agent's acceleration choice.

Model 1 Speed limit control, adapted from [26, Fig. 5]

spd	1	$\big(\quad (?v \leq v_{\text{des}}; a := *; ? - B \leq a \leq A)$
	2	$\quad \cup (?v \geq v_{\text{des}}; a := *; ? - B \leq a \leq 0)\big);$
atp	3	if $\left(e - p \leq \frac{v^2 - d^2}{2B} + \left(\frac{A}{B} + 1\right)\left(\frac{A}{2}\varepsilon^2 + v\varepsilon\right)\right)$ then $a := -B$ fi
drive	4	$t := 0;$
	5	$\{p' = v,\ v' = a,\ t' = 1\ \&\ v \geq 0 \wedge t \leq \varepsilon\}$

From Model 1, (Mitsch & Platzer, 2014) is used to obtain a ModelPlex formula, whose main components reflect the model structure:

$$
\begin{aligned}
\text{slc} \equiv \quad & 0 \le v \le v_{\text{des}} \wedge -B \le a^+ \le A \wedge e - p > S \\
& \vee\, 0 \le v_{\text{des}} \le v \wedge -B \le a^+ \le 0 \wedge e - p > S \\
& \vee\, 0 \le v \wedge a^+ = -B \wedge e - p \le S \qquad (1) \\
& \text{where } S \equiv \frac{v^2 - d^2}{2B} + \left(\frac{A}{B} + 1\right)\left(\frac{A}{2}\varepsilon^2 + v\varepsilon\right)
\end{aligned}
$$

Environment. LongiControl (Dohmen et al., 2021) provides a longitudinal vehicle control environment with state space $[\boldsymbol{x(t)}, \boldsymbol{v(t)}, \boldsymbol{a(t)}, \boldsymbol{a_{prev}(t)}, \boldsymbol{v_{lim}(t)}, \vec{\boldsymbol{v}}_{\boldsymbol{lim,fut}}\boldsymbol{(t)}, \vec{\boldsymbol{d}}_{\boldsymbol{lim,fut}}\boldsymbol{(t)}]$ of vehicle position $\boldsymbol{x(t)}$, speed $\boldsymbol{v(t)}$, acceleration $\boldsymbol{a(t)}$, previous acceleration $\boldsymbol{a_{prev}(t)}$, current speed limit $\boldsymbol{v_{lim}(t)}$, upcoming two speed limits $\vec{\boldsymbol{v}}_{\boldsymbol{lim,fut}}\boldsymbol{(t)}$ and distances to the upcoming two speed limits $\vec{\boldsymbol{d}}_{\boldsymbol{lim,fut}}\boldsymbol{(t)}$. The action space is continuous acceleration in the interval $\boldsymbol{[3, -3]m/s^2}$, and the sampling time is 0.1 s. The environment has posted speed limits of 50 km/h at [0, 250) m, 80 km/h at [250, 500) m, 40 km/h at [500, 750) m, and 50 km/h after 750 m. We map the environment to the ModelPlex formula (1) as follows: $\boldsymbol{x(t) \mapsto p, v(t) \mapsto v, a(t) \mapsto a^+, v_{lim,fut}(t)_1 \mapsto v_{des}, 3 \mapsto A, 3 \mapsto B, 0.1 \mapsto \varepsilon}$ with two separate configurations to demonstrate how the formal model can influence the behavior of the trained reinforcement learning agent:

Configuration 1 encourages behavior similar to Model 1, satisfying a speed limit before the speed limit begins: $\vec{\boldsymbol{v}}_{\boldsymbol{lim,fut}}\boldsymbol{(t)_1 \mapsto d}$, $\boldsymbol{x(t)} + \vec{\boldsymbol{d}}_{\boldsymbol{lim,fut}}\boldsymbol{(t)_1 \mapsto e}$, i.e., the first elements of the speed limit and speed limit position vectors are handed to the monitor.

Configuration 2 to illustrate the effectiveness of the monitor in influencing learned behavior, this configuration encourages "unusual" behavior opposing $\boldsymbol{r_{forward}}$ (2) by favoring meeting the posted speed limit by the end (rather than the beginning) of the speed limit: $\vec{\boldsymbol{v}}_{\boldsymbol{lim,fut}}\boldsymbol{(t)} \mapsto \boldsymbol{d}, \boldsymbol{x(t)} + \vec{\boldsymbol{d}}_{\boldsymbol{lim,fut}}\boldsymbol{(t)_1 \mapsto e}$.

Reward function. For evaluation, the reward function from (Dohmen et al., 2021, p. 1034) is used as a baseline, with weighted penalties for slow driving ($\boldsymbol{r_{forward}}$), energy consumption ($\boldsymbol{r_{energy}}$), jerk ($\boldsymbol{r_{jerk}}$), and speeding ($\boldsymbol{r_{safe}}$):

$$
\underbrace{-\xi_{\text{forward}} r_{\text{forward}} - \xi_{\text{energy}} r_{\text{energy}} - \xi_{\text{jerk}} r_{\text{jerk}}}_{\mathbf{R_G}} \underbrace{- \xi_{\text{safe}} r_{\text{safe}}}_{\mathbf{R_S}} \qquad (2)
$$

A safety robustness measure $\boldsymbol{Q(slc)}$ is obtained from formula (1), and then the speeding penalty is replaced with the (scaled) safety reward function as follows:

Logical constraint reward (LCR) with $\boldsymbol{R_S = \min\big(0, Q(slc)(s_0, s_1)\big) - 1}$ to ignore the safety robustness measure when satisfied and offset safety violations to exceed the largest magnitude of the $\boldsymbol{R_G}$ components in (2).

Logical constraint reward scaling (LCRS) applies the component-wise scaling of Example 4 to the safety robustness measure $\boldsymbol{Q(slc)}$.

Potential-based reward shaping (PBRS) applies the potential-based reward shaping of Example 5 to the safety robustness measure $\boldsymbol{Q(slc)}$.

Reward function LCR is used with Configuration 1 and reward functions LCR, LCRS, PBRS with Configuration 2.

Model training. Using these different reward functions, several agents are trained with the Soft Actor-Critic (SAC) method 1 and the hyperparameters of LongiControl (Dohmen et al., 2021, Table 4) with a learning rate of 1e-5. The baseline agent was trained with a learning rate of 1e-4.

Evaluation metrics. The training process was evaluated in terms of number of epochs until convergence, and the safety of the resulting agents in terms of the number and magnitude of speed limit violations. The team also quantified how successful the agents were in reaching the goal, which was to drive as fast and as close to the speed limit as possible, by measuring the accumulated reward per (2) and the difference between agent speed $\boldsymbol{v}$ and the posted speed limits $\boldsymbol{v_{des}}$ during an evaluation period. Note that training and evaluation episodes do not terminate early under unsafe behavior of the agent but instead invoke a negative reward as a penalty. Below are the results of the experiments.

Results. Figure 26 displays the average accumulated reward across several evaluation periods at every tenth epoch during training. Note that different reward functions are used during training, which means their magnitude is not directly comparable. For the baseline reward function, the reward converges to around −200 at 3000 epochs. The logical constraint reward using Configuration 1 converges to −300 at 1250 epochs. The following experiments using Configuration 2 converge to −325 at 1750 epochs for the logical constraint reward, −310 at 3000 epochs for logical constraint reward scaling, and −300 at 2250 epochs for potential-based reward shaping (faster or at the same rate as the baseline model). This helps with efficiency regarding how many epochs are required to finish training a reinforcement learning agent.

Figure 27(a) plots the accumulated reward according to the baseline reward function (2), so they can be compared relative to each other. The baseline agent has the highest accumulated reward, indicating that it operates more aggressively (i.e., less robustly) than other agents. The robustness nature of the other agents can also be seen in Figure 27(b), which plots the $\boldsymbol{v_{des} - v}$ to compare how well the agents achieve the goal of driving close to the speed limit.

When evaluating the agents for Configuration 1, compare LCR(1), logical constraint reward model against the baseline agent as their desired behavior is to drive as close to the speed limit as possible while satisfying the speed limit. After modifying the safety-oriented reward with the safety robustness measure from ModelPlex $\boldsymbol{Q(slc)}$, the resulting LCR(1) experiments resulted in agents that successfully satisfied the safety requirement by driving below the speed limit, while there were also some agents that violated safety when the upcoming speed limit was less than the current speed limit; using reward scaling to penalize unsafe behavior more can address this issue. The safe logical constraint reward agents had more robust behavior regarding maintaining safety as they left a larger gap to the desired speed limit, as seen in Figure 27(b).

For the agents using Configuration 2, they are generally more robust compared to the baseline agent by the increased distance to desired speed in Figure 27(b). Note that for Configuration 2, the purpose was to demonstrate how the formal model can influence the behavior of the learned agent, which is most prominent when emphasizing speed with LCRS(2), logical constraint reward scaling; the agent's behavior shows a preference for reaching the 80 km/h speed limit at the end of the speed limit at 500 m in Figure 27(b). This behavior extends into violating the monitor when passing into the subsequent 40 km/h speed limit.

The other Configuration 2 models, LCR(2) and PBRS(2), did not violate safety defined by the monitor.

In summary, the experiments suggest that training from modified reward functions LCR(1) converges faster, and LCR(2) and PBRS(2) generally satisfies safety. In addition, modifying aspects that correspond to the formal model can effectively change the agent's behavior LCRS(2), which can help with designing the reward function for different goals of reinforcement learning problems.

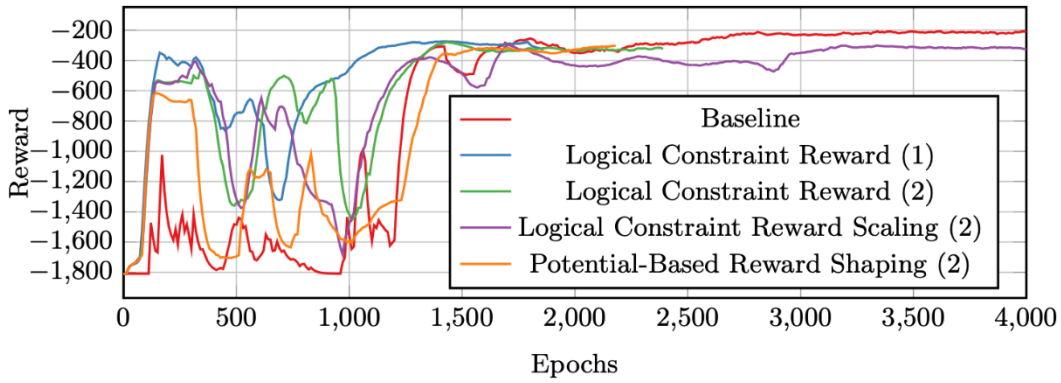
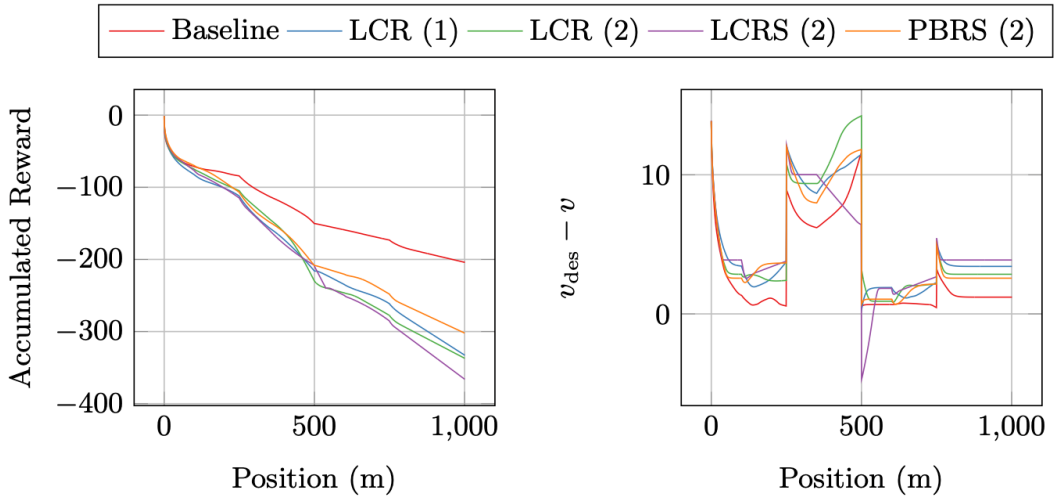


Figure 26. Accumulated reward across evaluations during training, smoothed with exponential moving average.



(a) Accumulated reward (b) Distance from speed limit

Figure 27. Averaged reward and performance across five evaluation runs at end of training.

8. Moving Block Controller Model

8.1 Moving Block Overview

The Moving Block (MB) concept in railway signaling refers to a method of train control and railroad traffic management that allows trains to operate more closely together on a track. Traditional fixed-block signaling divides the track into fixed sections, and only one train is allowed in each section at a time, maintaining a safe distance between them. In contrast, the moving block system dynamically adjusts the safe separation between trains based on real-time factors.

The system continuously monitors the position, speed, and other relevant parameters of trains using advanced signaling and train control technologies. Based on these relevant parameters, the Moving Block Controller (MBC) calculates a dynamic safe separation distance between trains based on their current speeds, braking capabilities, and other factors. This allows trains to operate more efficiently while maintaining safety. The trains communicate with a dispatching system, and the MB control system communicates with all trains in the network. This enables real-time adjustments to the safe separation distance and allows for precise control of train movements.

Trains equipped with the MB system receive speed commands from the dispatching system. These commands are continuously adjusted to ensure that trains maintain required safe separation. By allowing trains to operate more closely together while ensuring safety, the MB system increases the overall capacity of the railway network. This can lead to improved efficiency, reduced travel times, and increased flexibility in scheduling.

The MB system includes collision avoidance mechanisms to prevent trains from getting too close to each other. If a potential conflict is detected, the system can even initiate emergency braking or take corrective actions to avert a collision.

Overall, the MB concept represents a shift from fixed, conservative block signaling to a more dynamic and adaptable approach, optimizing railway operations while maintaining safety standards.

8.2 Formal Moving Block Model

An MBC was modeled and verified in this research. As in Section 3 and Section 4, MBC behavior was formally modeled in differential dynamic logic, deliberately including minimal implementation details and limiting the specification to an interface that is relevant to safety. This resulted in a control envelope permitting many specialized concrete implementations (including a machine learning based controller). Removing unnecessary complexity stemming from implementation details that do not impact safety also reduces the complexity of the proof.

Model 6 shows that the MBC is modeled as optionally assigning a new EOA at the start of every control cycle (Line 2). The only restriction on the assignment is that the new EOA cannot be so close that even when immediately employing maximum braking, the train would be unable to stop in time. The reason that the assignment is modeled to be at the start of a control cycle is that even if a message reassigning the motion authority is received in the middle of a control cycle, the controller still can only react to it at the start of the next control cycle. As a result, controller behavior is identical to when motion authority messages are in fact received only at the start of

control cycles. Situations like network failures, where the train does not receive a new motion authority, are modeled by Line 3 of Model 6, which keeps the previous MA unmodified. The rest of the model shows the train controller and dynamics. It remains the same as Model 4.

Model 6 Moving block controller and dynamics.

Reset Timer	1	$t := 0;$
MBC	2	$\big(\quad e := *; ?\texttt{brakeDist}_a(v, \max a_{b\max}, a_b)$
	3	$\cup \quad ?\text{true}\big);$
Free driving Braking Dynamics	4	Model 4

8.3 Safety Proof Moving Block Model

The team formally verified that an MBC following the restriction of Model 6, Line 2 is safe.

Theorem 5. *The moving block controller guarantees that a train (running the controller of Model 4) stays within its MA. The* dL *formula below is provable.*

$$\text{init}_{\boldsymbol{s}}(\text{brakeDist}_{\boldsymbol{a}}) \land \text{initAirbrake} \rightarrow [(\text{Model } \mathbf{6})^*]\boldsymbol{e - p > 0}$$

Proof. By proof in KeYmaera X. The proof of this theorem is in principle very similar to that of Theorem 4 but is twice the size because of the extra case where the EOA is reassigned. The same loop invariant as Theorem 4 is maintained (using dL rule loop). Then, the case where the motion authority is unchanged is the same as Theorem 4. When the motion authority is reassigned, despite syntactic differences, the same chain of reasoning still works because the crucial property of the motion authority that made safety possible is still ensured by the test condition of Line 2.

The model presented here identifies conditions on an MBC crucial to *safety.* But it is important for an MBC also to have *liveness,* which ensures that trains make progress and reach their desired destination, never entering a deadlock. Identifying conditions that provably ensure liveness is an interesting avenue for future work. Instead of mandating a *minimum* authorized distance ahead as Model 6 does for train safety, ensuring liveness will likely involve conditions on *maximum* motion authority extensions so that one train does not monopolize more track than necessary and stop other trains from moving.

9. Moving Block Controller Learning

This section describes MBC learning in greater detail.

9.1 Moving Block Architecture Blueprint

Figure 28 serves as a blueprint architecture for MB systems as described in Section 10.1. The Locomotive Systems block consists of an Onboard Unit (OBU) that computes the train location using other systems in the locomotive (e.g., GPS) and continuously communicates back to the MB server. The Back Office Server provides train consists, routes, initialization data, movement authorities and other data to the OBU and synchronizes subdivision data with the dispatching system. The dispatching system monitors train movements by integrating information from the OBU and Back Office Server. It sends restrictions and provides movement authorities to the Back Office Server. The MB Server dynamically adjusts the separation distance between the trains based on real-time data, routes, and other parameters. Crucially, it includes a *function* "Moving Block Calculator" that describes an interface for computing movement authorities, which can be implemented using traditional engineering methods as well as machine learning.

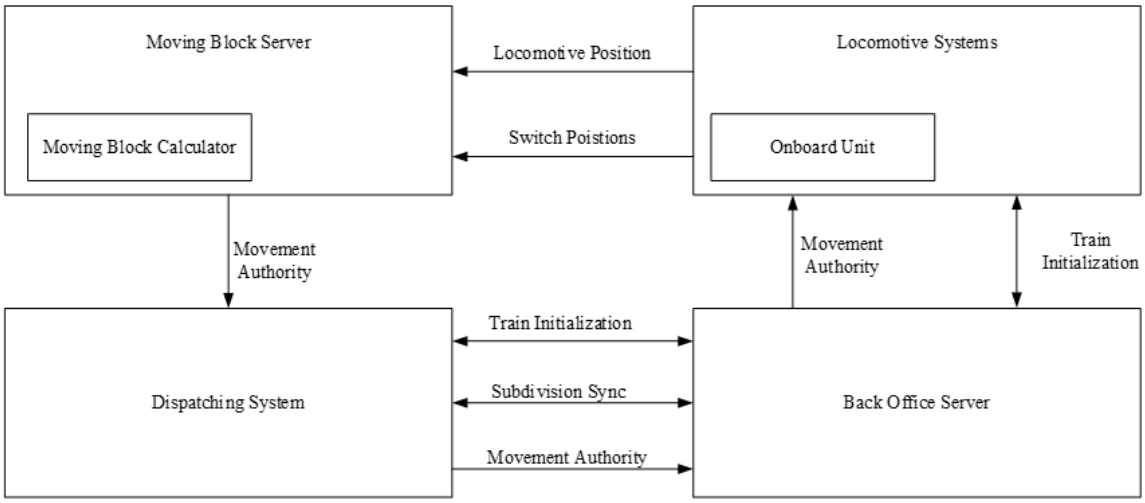


Figure 28. Moving block architecture blueprint.

9.2 Moving Block Control Objectives

For implementing the MB Calculator function, reinforcement learning is used with a reward function that captures performance and safety objectives. The verified safety envelope of Model 6 can be used to provide provable safety guarantees about the learned controller at runtime.

Evaluating the effectiveness of a learned MBC involves measuring various metrics to ensure that the system meets operational goals and safety standards. While metrics for a final operation should focus on indicators such as the number of trains that can pass through a specific section of the track within a given time, the average time it takes for a train to travel between two designated points, or the percentage of time a train occupies a specific track section, the focus here is on more local safety and performance objectives for learning that can be measured at each time point during learning.

Safety Metrics: The separation of moving blocks is the most important safety property to avoid collisions between trains.

- Determine overlapping movement authorities when using MBC. In train routing scenarios (where a train is moved to a siding) assigned movement authorities should not overlap between tracks.

Performance Metrics:

- Distance between trains when a train is following another train, so that the movement authorities are as close to each other as possible
- Average speed the train will run within the authority speed limits – difference to posted speed limit
- The speed at which the moving block controller can adjust to changes in train speeds, routes, or unexpected events – number of movement authorities that can be sent
- Threshold to obey with the braking curve stopping distance (i.e., always less than a certain factor of the engineered stopping distance)

Further metrics can be defined that evaluate the learning and adaptability of the MBC. These include assessing how quickly the MBC adapts to changing conditions and improves its performance over time. The convergence of the learning algorithms and their stability in different scenarios also is measured.

These safety and performance objectives collectively provide a comprehensive view of the MBC's effectiveness in terms of operational safety, performance, and operational and learning performance.

9.3 Learning Scenarios

The team selected four real-world train scenarios for controller learning, consisting of trains operating on a track in the same or opposite directions.

Scenario 1. Two equipped trains meet in opposite directions (Safety).

In this scenario the MB generates automatic MA updates which do not overlap the limits of an existing track block (Figure 29).

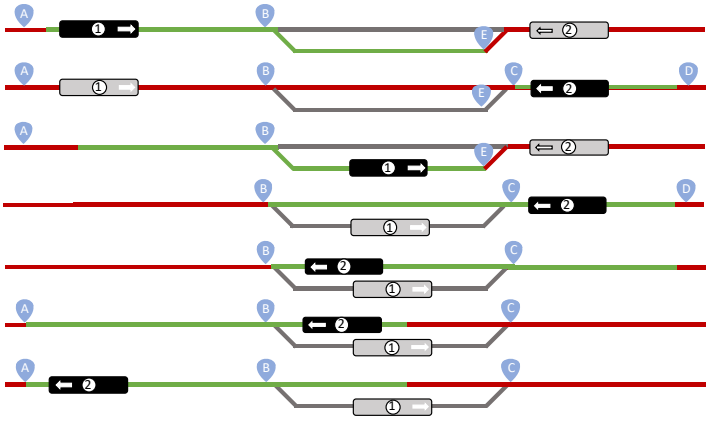


Figure 29. Moving Block Scenario 1.

As depicted, the dispatcher wants to place a track block to prevent the controller from extending an MA onto a given track. The block is placed on a track that is not occupied by a train (i.e., no existing train authority is there).

Scenario 2. Two equipped trains meet in opposite directions (Safety and Performance).

Scenario 2 has two goals of showing (a) the safety that the generated authority updates will not overlap any limits of an existing track block and (b) its performance on routing one train in a siding to allow the other train to continue along the main track (see Figure 30). Two equipped trains meet in opposing directions. While one train is routed to a siding, the other train is routed on the main track.

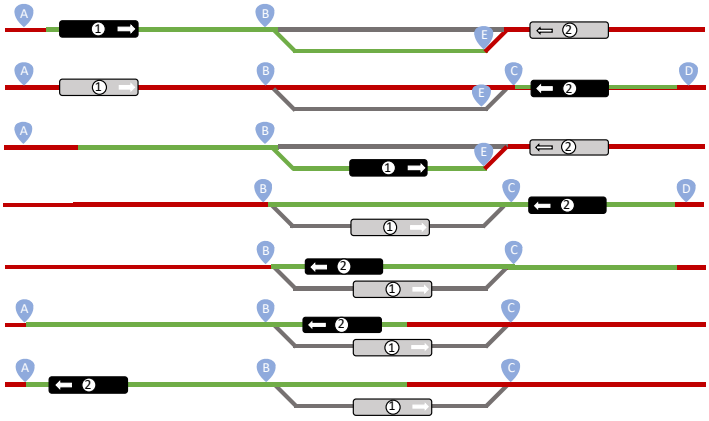


Figure 30. Moving Block Scenario 2.

As the trains approach each other, MB should ensure that no overlapping MA are issued. When Train 2 is moved into the siding and frees up the main track, MB should issue a new MA to Train 2 to continue. As the MA of Train 2 is rolled up, MB should issue a MA to Train 1 to continue along the main track.

Scenario 3. An equipped Train 1 is followed by an equipped Train 2 and routed through siding.

An equipped Train 1 is followed by an equipped Train 2. Train 1 passes over the switch, but Train 2 takes a diverging route through the switch (see Figure 31).

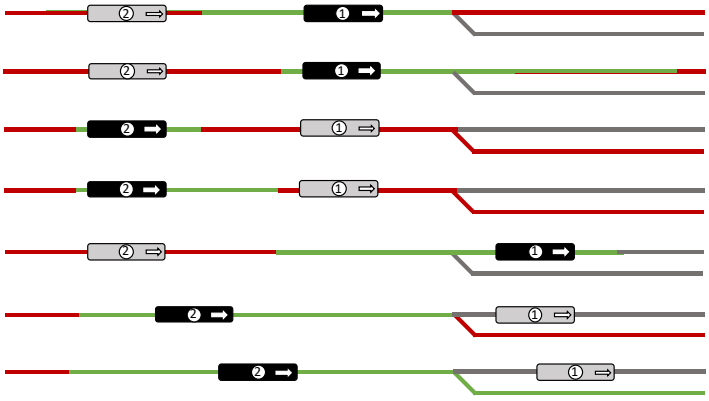


Figure 31. Moving Block Scenario 3.

MBC generates automatic movement authorities for the 2 trains, one following the other closely (performance) but safely (safety), so there is no overlap, allowing each train to take a different route over main.

Scenario 4: An equipped Train 1 is followed by an equipped Train 2

Train 1 is following Train 2, with slower speed than Train 1 (see Figure 32).

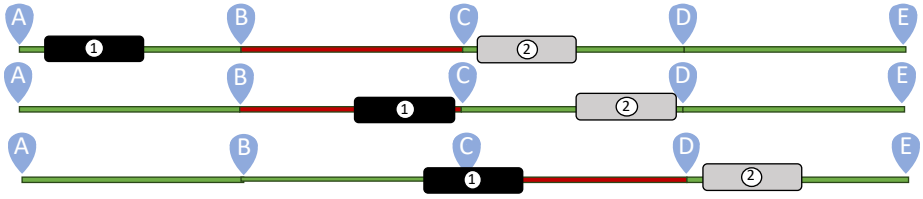


Figure 32. Moving Block Scenario 4.

Scenario 4 is a simple MB scenario which was selected to evaluate a minimal viable safety case. The two trains run in the same direction and operate in proximity defined by the MB. Train 2 is the lead train, with Train 1 following behind. The MB system determines an initial safe separation distance between the two trains based on their speed, braking capabilities, and other factors.

As Train 2 accelerates or decelerates, the MB system adjusts dynamically for the safe separation distance to maintain a suitable safety gap between the trains.

If Train 2 slows down, the system allows Train 1 to approach more closely, optimizing the use of the track space. Both trains continue moving along the track adjusting speeds and separation distances as needed.

The key aspect of this scenario is the continuous adaptation of a safe separation distance allowing the trains to operate more closely together while maintaining safety.

9.4 Moving Block Simulator and Moving Block Learning

The team used reinforcement learning to create a controller that runs from within the verified control envelope of Model 6 by providing the reassignments of Line 3. Reinforcement learning can permit the encoding of clever optimizations to save energy or increase throughput into the procedure that reassigns motion authorities, while the guard condition of Line 3 still ensures safety.

There are two ways to implement the MBC: in a *centralized* MBC, a single procedure reassigns the motion authorities of all trains at once, while in a *decentralized* design, many local procedures, one for each train, compute and reassign the MA for a single train using available information like the current speed, position, and desired route, or the current MAs of the other trains. Both designs have advantages and drawbacks. The centralized MBC has greater freedom to coordinate reassignments, potentially leading to more efficient motion planning with higher train throughput. The decentralized design transforms a planning problem into a control problem (known to be easier to solve with reinforcement learning) and is potentially a more fault-tolerant paradigm since all trains do not need to rely on correct communication with a single MBC. The team created a reinforcement learning environment for both designs. All the environments are

written in Python with OpenAI gym (Brockman, et al., 2016). Training is performed using algorithms from the stable baselines 3 library (Raffin, et al., 2021).

9.4.1 Centralized MBC

The central MBC environment models Scenario 2 (i.e., trains from opposing directions have a siding). The *observations* that the learnt controller may use to decide MA assignments consists of 1) each train's position and velocity, both of which are floating point numbers, 2) each train's MA, where MA is represented by its starting and ending block, and 3) switch positions, where each switch is either *locked* in standard or reverse position depending on which track it shunts the train to, or *indeterminate* when the switch is not locked and should be avoided. The learnt agent representing the MBC responds in an *action space* consisting of new MA assignments for each train. Based on these MAs, the environment then determines train motion by a fixed motion algorithm based on Model 1.

The learning reward function has many components. A goal penalty penalizes trains for being away from the target destination and for traveling too slowly. An MA penalty penalizes trains for exceeding their MA, encouraging the learnt MBC agent to provide reasonable MAs. There is a large negative reward when MAs intersect. Termination happens when either the trains collide or complete traveling the section of the track that the environment models. There is a positive terminal reward for route completion without collision and negative reward on collision. Training was performed over 10,000 steps using the PPO, A2C, and TRPO algorithms but failed to produce effective results. The team investigated the simpler Scenario 4 where two trains follow each other. Figure 33 shows the learning environment visualization of Scenario 4 where the bright blue blocks and bright red blocks each represent the two trains, while the dark blue and dark red blocks represent their MAs. The blue train follows the red train. The observation and action spaces remain the same in this new scenario (though observation information about track switch positions is no longer relevant).

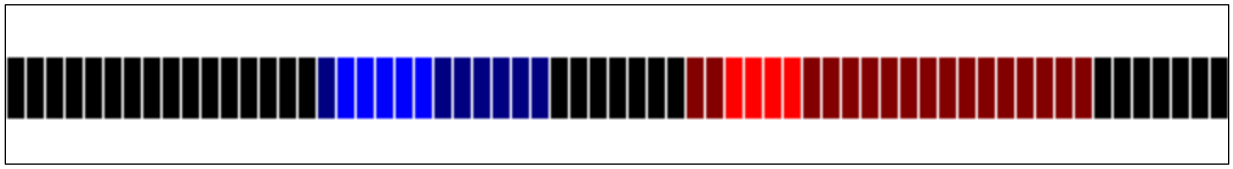


Figure 33. Visualization of Scenario 4 in the learning environment.

Experiments were run with learning rates 0.01, 0.001, and 0.0001, and learning was found to be most effective at rate 0.001. Figure 34 shows the impact of learning rate. The y-axis shows mean reward, while the x-axis shows number of time steps of training. The purple curve is produced by a learning rate of 0.001 while the yellow learning curve is produced by a learning rate of 0.01. Learning over one million timesteps produced policies that correctly assigned MAs around trains that allowed some forward motion, but the strategies were still suboptimal and did not maximize train speeds or rewards. The planning nature of reassigning multiple MAs at once may pose a limiting factor that requires additional training and/or additional methods. Future research may produce better results with further reward shaping and hyperparameter tuning.

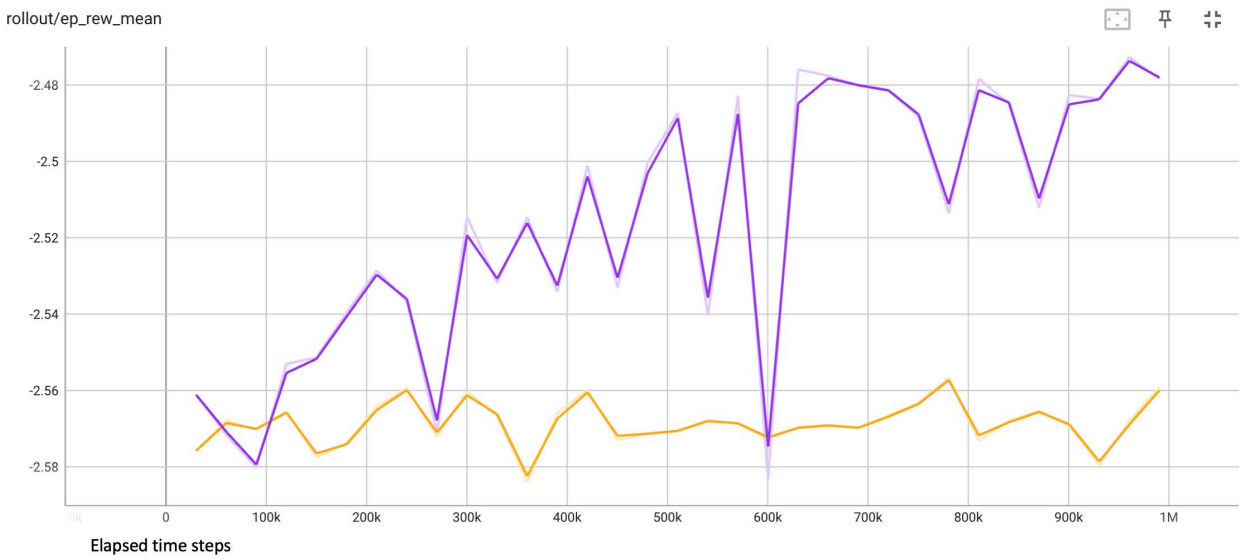


Figure 34. Impact of learning rate on mean reward.

9.4.2 Decentralized MBC

The team also created a training environment for the decentralized MBC design under Scenario 4. The observations that the learnt controller may use to decide MA assignments consist of 1) the distance (as computed by a train's onboard unit) that the train will take to brake and 2) the distance from the train's current position to the next position along the track that is assigned to some other train's MA. The actions that the learnt MBC provides consist of a single number that indicates the length by which to extend the train's MA (see Figure 35). As before, based on the new motion authorities, the environment determines train motion.

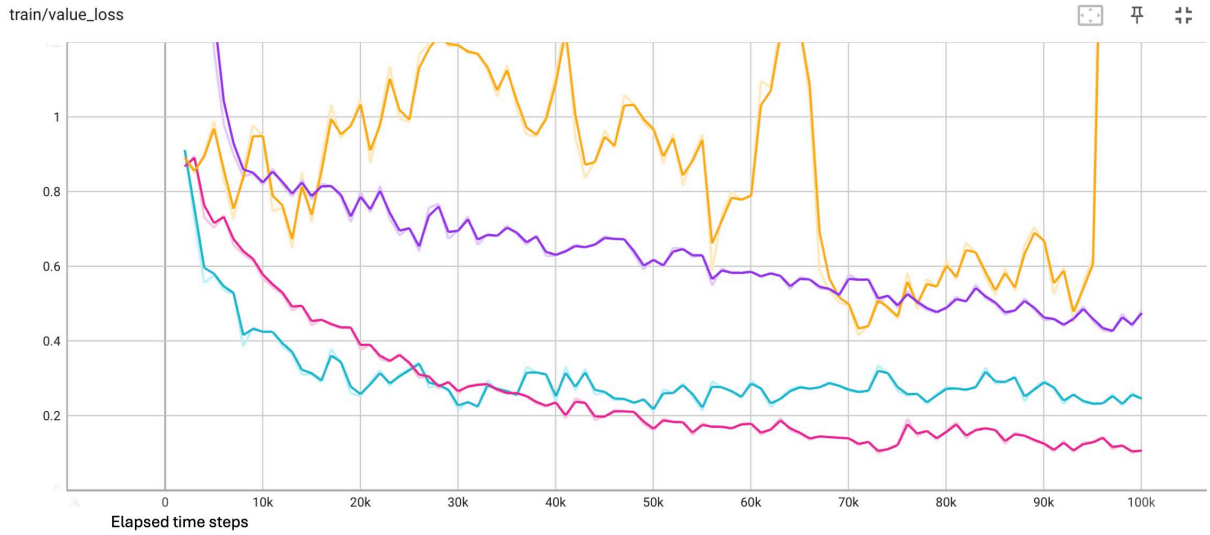


Figure 35. Impact of learning rate on loss.

The learning reward function is the same as the centralized MBC. Termination again happens when either the trains collide or complete traveling the section of the track in the environment. Training was performed over 100,000 steps using the PPO algorithm. The resulting policy

outperformed a handwritten baseline policy by obtaining a higher mean reward. In the baseline policy, the lead train's MA is extended by a constant amount, and the following train is assigned an MA up to the lead train.

Experiments were run with learning rates 0.01, 0.001, 0.0001, and 0.00001, and learning was found to be most effective at rate 0.0001. Figure 35 shows the impact of learning rate. The y-axis shows loss, while the x-axis shows the number of training time steps. The yellow curve is produced by a learning rate of 0.01, the blue curve is produced by a learning rate of 0.001, the pink curve is produced by a learning rate of 0.0001, and the purple curve is produced by a learning rate of 0.00001.

9.5 Moving Block Execution Environment

The team employed simulation tools to develop a virtual environment that mimics real-world railway operations to test the learned MBC. The MB test environment allows testing the MBC in various scenarios without risking safety. While it is crucial to ensure that the simulator includes a diverse set of scenarios (e.g., normal operations, emergency situations, and potential failure modes), the team considered a selection of scenarios. As a future use of the simulation environment, it allows for integration with the Siemens MBC implementation for direct comparison between a learned controller and an engineered controller. An additional future benchmark opportunity is comparison against traditional fixed-block signaling systems.

The MB Execution environment is a proof of concept that tests the MBC and can run various real-world train scenarios in a lab environment. It also demonstrates how MB is integrated into a Positive Train Control (PTC) system. The building blocks of the MB Execution Environment are discussed in the next section.

9.5.1 Building Blocks of MB Execution Environment

The Siemens MB Execution environment consists of a combination of real-world and simulated components: Locomotive Systems Simulation, Back-Office Server (BOS), Dispatching Simulation, and MB Server (see Figure 36).

- Locomotive Systems Simulation. An OBU running in a simulated environment controls the train environment allowing to start a train from a given position and with a given speed. It further controls the Human Machine Interface (HMI) of the OBU which visualizes the train behavior. In addition, it includes other simulators to support the overall locomotive simulation, such as the wayside interface unit (WIU) and GPS simulator.
- Back Office Server. The BOS enables interaction between the Dispatching System and the OBU. It synchronizes subdivisions with Dispatch Simulation and provides train initialization and movement authorities to the OBU.
- Dispatching Simulation. This simulates all the basic functions of a dispatching system, primarily issuing MAs to the BOS.
- Moving Block Server. This includes all the minimum required functionalities of MB: MB calculation, train tracking based on positions, extend or roll-up authorities, conflict checking, etc. The MB Calculator is a module that is responsible for managing train movements by calculating the movement authorities based on train positions and routes.

The MB calculation is based on several factors, such as the train's warning distance (i.e., the distance required for a train to come to a complete stop), speed, and other parameters (e.g., reaction times, minimum MA lengths, etc.).

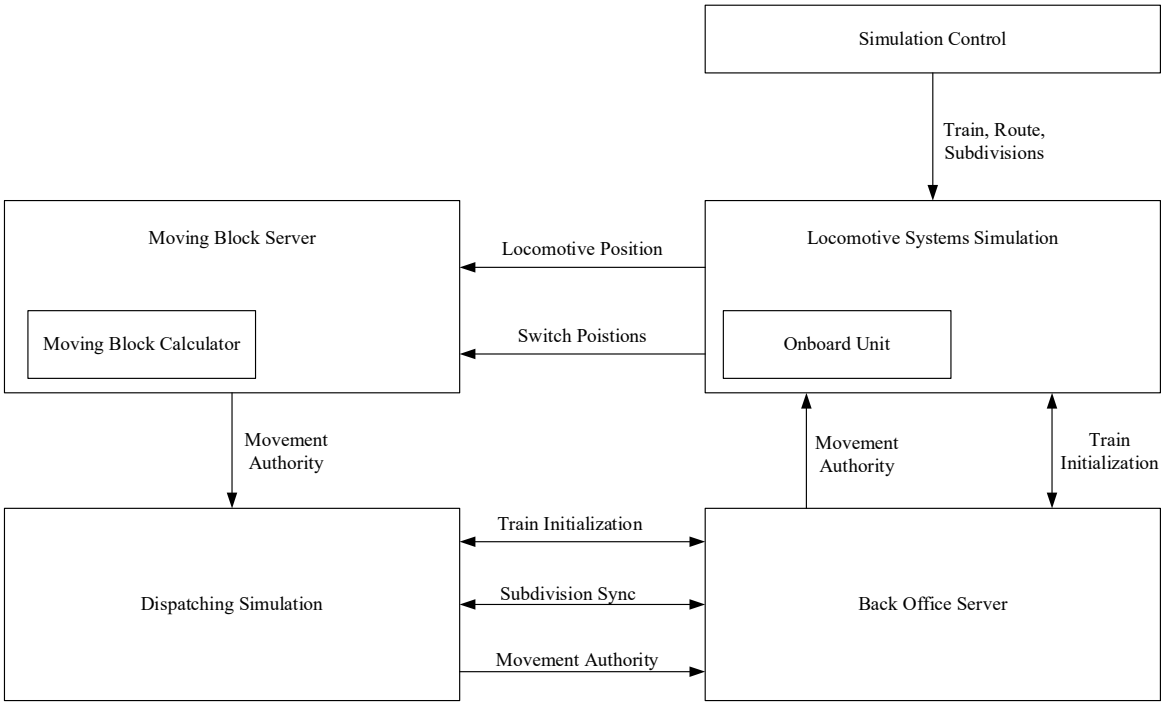


Figure 36. MB Execution Environment.

9.5.2 Simulation Flow

The Simulation Flow can be explained in four steps:

1. **Initialization:** The Simulation Control provides initial data such as the train, route and subdivisions to the Locomotive Systems Simulation which is forwarded to the Back Office Server. The BOS synchronizes the subdivisions and initializes the trains.
2. **Localization**: The Onboard Unit sends train positions to the MB Server.
3. **Starting of trains:** The MB Server sends MA for the trains.
4. **Train travelling.** The OBU sends updated position reports and switch positions to the MB Server, the MB Calculator calculates the needed MA, and the MB Server sends this to the Dispatching Simulation. Any authorized MAs provided by the BOS sent to the OBU is used to update the locomotive position for the MB Server.

9.6 Sim-To-Real Transition

The team combined the verified safety envelope of the formal model with reinforcement learning. A machine learning trained MBC should be capable of issuing MAs without compromising safety and performance. Testing in the simulation environment used for training is a first step toward demonstrating the correctness of a learned controller; the remaining task is to transfer the learned controller into the real-world execution environment of Section 11.5 (based on the MB blueprint architecture). The execution environment provides the ability to execute

real-world scenarios. The engineering challenges that come with transferring a learned controller into an execution environment are discussed below.

9.6.1 Integration of MB Learned Controller in the Siemens MB Execution Environment

Because of programming language differences, the learned controller in Python must be adapted to the execution environment language C. To this end, Python code is cross-compiled to a C library that can be statically linked with the execution environment. The internal logic of the MB Calculator in the execution environment is replaced with function calls to the learned MBC C functions. Figure 37 below shows that the MB Calculator module within MBS will be replaced by function calls to the learned MB Controller.

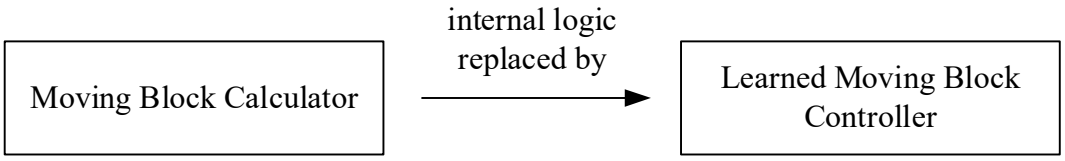


Figure 37. MB Environment integration.

The inputs and outputs of the learned controller consist of primitive data types whereas the inputs and outputs of the MBC consist of complex data structures. Figure 38 illustrates this datatype conversion.

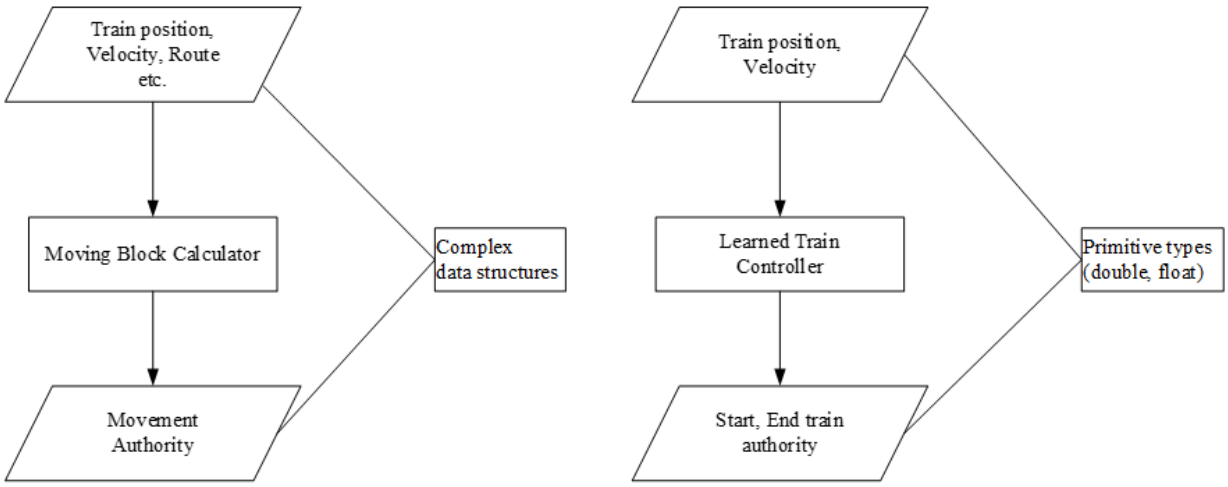


Figure 38. Input and output types of Siemens MBC vs. Learned Train Controller.

Summary. Integration of the Learned MBC in the execution environment requires the following three steps:

1. Convert MB Calculator complex data structure inputs to primitive data types which the Learned Controller accepts.
    - Train Position and Velocity: Train position can be derived from block positions (head and rear offsets). The mid-point of a block can be computed as the train's position. Velocity is available which can be converted to primitive data type.

- Route: The route information can be used to determine relevant sections of the track where the train is located or to determine the direction of the train.

2. Call the Learned Controller C functions to compute the start and end train authorities (primitive data types).
3. Convert the learned controller primitive data type outputs (e.g., start, end authority) to the expected complex data structures of the MB Calculator. Based on the authority start and authority end, the complex data structure that represents the authorized path for the train can be constructed. This involves identifying the specific blocks and switches in the route that fall within the start and end authority range.

Validation. The MB Execution Environment supports two modes for validation: one with the Siemens MB Calculator logic and the other with the learned controller logic. The test scenarios are executed in these two modes and logs can be collected for comparison and future analysis.

10. Conclusion

The research team created formally verified train controllers consisting of a kinematics model with all its competing influences of track grade, curve resistance, air brakes, and Davis resistance. Techniques that generalize to resolve challenges in safety critical embedded software design improved controller efficiency. Simulation on the kinematics model Eq. (2) shows significant improvement in undershoot over conservative controllers that use safety offsets, and improved safety compared to controllers without safety offsets. In testing-based parametrization of the verified model using the TOES simulator, the team studied how to obtain model instances of the formal freight train models that can predict (simulated) train behavior over a wide variety of train configurations. The outcome of the validation showed that, on average, the parameters based on (Brosseau & Ede, 2009) and (Brosseau J. , Ede, Pate, Wiley, & Drapa, 2013) of the freight train model align well with the simulator. Techniques to couple formal models with reinforcement learning can be used to then turn formal models into reward functions to support the learning process itself and obtain agents that behave robustly inside the formally verified safety envelope.

11. References

Berger, U. et al. (2018). Verification of the European Rail Traffic Management System in Real-Time Maude. *Science of Computer Programming,* Volume 154, pp. 61-88.

Bonacchi, A. & Fantechi, A. (2014). *On the Validation of an Interlocking System by Model-Checking.* s.l., s.n., pp. 94-108.

Brockman, G. et al. (2016). Openai gym. *arXiv preprint arXiv:1606.01540.*

Brosseau, J. & Ede, B. M. (2009). *Development of an Adaptive Predictive Braking Enforcement Algorithm,* s.l.: s.n.

Brosseau, J. et al. (2013). *Development of an Operationally Efficient PTC Braking Enforcement Algorithm for Freight Trains,* s.l.: s.n.

Budnik, C., Gario, M., Markov, G. & Wang, Z. (2018). *Guided test case generation through AI enabled output space exploration.* s.l., s.n., pp. 53-56.

Cimatti, A. et al. (2012). *Formal Verification and Validation of ERTMS Industrial Railway Train Spacing System.* s.l., s.n., p. 378–393.

Claudio, M. D. et al. (2014). *Model-based development of an automatic train operation component for communication based train control.* s.l., s.n., p. 1015–1020.

Dasher, J. & Morrison, K. (2020). *Evaluation of PTC Braking Enforcement Algorithms for Passenger and Commuter Trains,* s.l.: s.n.

Davenport, J. H. & Heintz, J. (1988). Real Quantifier Elimination is Doubly Exponential.. *J. Symb. Comput.,* Volume 5, pp. 29-35.

Dohmen, J., Liessner, R., Friebel, C., & Baker, B. (2021). LongiControl: A Reinforcement Learning Environment for Longitudinal Vehicle Control. ICAART (2), pp. 1030-1037.

Donze, A., Ferrere, T., & Maler, O. (2013). Efficient Robust Monitoring for STL. *CAV 2013*, pp. 264-279.

Eckert, J. et al. (2021). A fast simulation approach to assess draft gear loads for heavy haul trains during braking. *Mechanics Based Design of Structures and Machines.*

Essamé, D. & Dollé, D. (2006). *B in Large-Scale Projects: The Canarsie Line CBTC Experience.* s.l., s.n., pp. 252-254.

Fainekos, G., & Pappas, G. (2006). Robustness of Temporal Logic Specifications. FATES/RV 2006, pp. 178-192.

Fainekos, G., & Pappas, G. (2009). Robustness of Temporal Logic Specifications for Continuous-Time Signals. *Theor. Comput. Sci., 410*(42), pp. 4262-4291.

Fantechi, A. (2019). *Connected or Autonomous Trains?.* s.l., s.n., p. 3–19.

Ferrari, A. et al. (2019). *Survey on Formal Methods and Tools in Railways: The ASTRail Approach.* s.l., s.n.

Fulton, N. et al. (2015). *KeYmaera X: An Axiomatic Tactical Theorem Prover for Hybrid Systems.* s.l., s.n., pp. 527-538.

Fulton, N. & Platzer, A. (2018). *Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning. AAAI 2018: 6485-6492.* s.l., s.n.

Fulton, N. & Platzer, A. (2019). Verifiably Safe Off-Model Reinforcement Learning. *TACAS* (1), pp. 413-430.

Gerhart, S. L., Craigen, D. & Ralston, T. (1994). Case study: Paris Metro Signaling System. *IEEE Software,* Volume 11, pp. 32-28.

Hay, W. W. (1982). *Railroad engineering.* 2nd ed. ed. New York: Wiley.

Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., Wojtczak, D. (2020). Faithful and Effective Reward Schemes for Model-Free Reinforcement Learning of Omega-Regular Objectives. *ATVA 2020*, pp. 108-124.

Hammond, L., Abate, A., Gutierrez, J., & Wooldridge, M. (2021). Multi-Agent Reinforcement Learning with Temporal Logic Specifications. *AAMAS 2021*, pp. 583-592.

Hasanbeig, M., Abate, A., & Kroening, D. (2020). Cautious Reinforcement Learning with Logical Constraints. *AAMAS 2020*, pp. 483-491.

Kabra, A., Laurent, J., Mitsch, S. & Platzer, A. (2024). *CESAR: Control Envelope Synthesis via Angelic Refinements.* s.l., s.n.

Kabra, A., Mitsch, S. & Platzer, A. (2022). Verified Train Controllers for the Federal Railroad Administration Train Kinematics Model: Balancing Competing Brake and Track Forces. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* pp. 1-1.

Kamburjan, E. & Hähnle, R. (2017). *Deductive Verification of Railway Operations.* s.l., s.n., p. 131–147.

Karra, S. L., Larsen, K. G., Lorber, F. & Srba, J. (2019). *Safe and Time-Optimal Control for Railway Games.* s.l., s.n., p. 106–122.

Konighofer, B., Lorber, F., Jansen, N., & Bloem, R. (2020). Shield Synthesis for Reinforcement Learning. ISoLA (1), pp. 290-306.

Loos, S. M. & Platzer, A. (2016). *Differential Refinement Logic.* s.l., s.n., pp. 505-514.

Mitsch, S. et al. (2017). *Formal Verification of Train Control with Air Pressure Brakes.* s.l., s.n., pp. 173-191.

Mitsch, S. & Platzer, A. (2011). The Structure of Differential Invariants and Differential Cut Elimination. *Logical Methods in Computer Science*, *8*(4).

Mitsch, S. & Platzer, A. (2014). *ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Models.* s.l., s.n., pp. 199-214.

Ng., A., Harada, D., & Russell, S. (n.d.). Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. *ICML 1999*, pp. 278-287.

Platzer, A. (2008). Differential Dynamic Logic for Hybrid Systems.. *J. Autom. Reas.,* Volume 41, pp. 143-189.

Platzer, A. (2010). *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics.* Heidelberg: Springer.

Platzer, A. (2017). A Complete Uniform Substitution Calculus for Differential Dynamic Logic. *J. Autom. Reas.,* Volume 59, pp. 219-265.

Platzer, A. (2018). *Logical Foundations of Cyber-Physical Systems.* s.l.:Springer.

Platzer, A. & Quesel, J.-D. (2009). *European Train Control System: A Case Study in Formal Verification,* s.l.: s.n.

Qian, M., & Mitsch, S. (2023). Reward Shaping from Hybrid Systems Models in Reinforcement Learning. NFM 2023, pp. 122-139.

Raffin, A. et al. (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research.*

Richardson, D. (1968). Some Undecidable Problems Involving Elementary Functions of a Real Variable. *J. Symb. Log.,* Volume 33, pp. 514-520.

Teodoro, Í. P. et al. (2020). Parallel simulation of railway pneumatic brake using openMP. *International Journal of Rail Transportation,* April, Volume 8, p. 180–194.

Tran, H-D. et al. (2019). Safety Verification of Cyber-Physical Systems with Reinforcement Learning Control. *ACM Trans. Embed. Comput. Syst., 18*(5s).

Vu, L. H., Haxthausen, A. E. & Peleska, J. (2017). Formal modelling and verification of interlocking systems featuring sequential release. *Science of Computer Programming,* Volume 133, pp. 91-115.

Wahl, M. (2010). *Survey of railway embedded network solutions - Towards the use of Industrial Ethernet technologies. In: Les Collections de l'Inrets, Synthése S61, 104 pages.* s.l.:s.n.

Weispfenning, V. (1988). The Complexity of Linear Problems in Fields. *J. Symb. Comput.,* Volume 5, pp. 3-27.

Zhang, Z. et al. (2021). On the Effectiveness of Signal Rescaling in Hybrid System Falsification. NFM 2021, pp. 392-399.

Zou, L. et al. (2014). *Verifying Chinese Train Control System under a Combined Scenario by Theorem Proving.* s.l., s.n., p. 262–280.

Appendix A.
Kinematic Train Motion Model

Instantiates Model 1 and Model 2.
Corollary 6 (Physical train model is safe). *The detailed train motion never overshoots the EOA:*

$$\text{init} \rightarrow [(\text{Model 6})^*]\boldsymbol{e} - \boldsymbol{p} > \mathbf{0}$$

where stopDist *is either expanded according to Eq.* (4) *or* (8)*.*

Proof. By uniform substitution from for Theorem 1 for (4) or from Theorem 2 for (8), using the substitutions σ below:

$$\sigma = \begin{cases} a_{max} \mapsto \dfrac{F_L}{m_T} - \dfrac{A_R w + B_R n}{m_T} \\ a_s(p) \mapsto -\dfrac{A_G w \cdot grade(p)}{m_T} \\ a_r(v) \mapsto -\dfrac{C_R w v + D_R v^2}{m_T} \\ b_{max} \mapsto \dfrac{F_B}{m_T} + \dfrac{A_R w + B_R n}{m_T} \\ a_{c(p)} \mapsto -\dfrac{A_C w \cdot curve(p)}{m_T} \end{cases}$$

Model 6 Kinematic train controllers in FRA motion model.

Reset timer

1 $t := 0;$

Free driving

2 $\Big($ $?e - p > \texttt{stopDist}(p, v);\ a_l := *;$

3 $?\left(\begin{array}{l} -\dfrac{F_B}{m_T} - \dfrac{A_R \cdot w + B_R \cdot n}{m_T} \\ \quad \leq a_l < \dfrac{F_L}{m_T} - \dfrac{A_R \cdot w + B_R \cdot n}{m_T} \end{array}\right);$

4 $a_b := 0;\ m_b := 0$

Braking

5 $\cup\ a_l := -\dfrac{F_B}{m_T} - \dfrac{A_R \cdot w + B_R \cdot n}{m_T};$

6 $m_b := m_p\Big);$

Dynamics

7 $\{p' = v,$

8 $v' = a_l - \dfrac{A_G \cdot w \cdot \mathit{grade}(p)}{m_T}$

9 $- \dfrac{A_C \cdot w \cdot \mathit{curve}(p)}{m_T}$

10 $- \dfrac{C_R \cdot w \cdot v + D_R \cdot v^2}{m_T},$

11 $t' = 1\ \&\ t \leq T \wedge v \geq 0\}$

Abbreviations and Acronyms

ACRONYM	DEFINITION
BOS	Back-Office Server
dC	Differential Cut
dI	Differential Invariant
dL	Differential Dynamic Logic
EOA	End of Movement Authority
FRA	Federal Railroad Administration
HMI	Human Machine Interface
MA	Movement Authority
MB	Moving Block
MBC	Moving Block Controller
OBU	Onboard Unit
ODE	Ordinary Differential Equation
SAC	Soft Actor-Critic
TOES	Train Energy and Operations
WIU	Wayside Interface Unit