Formally Verified Safety Net for Waypoint Navigation Neural Network Controllers *

Alexei Kopylov[1], Stefan Mitsch[2], Aleksey Nogin[1], and Michael Warren[1]

[1] HRL Laboratories, LLC, Malibu, CA
{akopylov, anogin, mawarren}@hrl.com
https://csrs.hrl.com/
[2] Carnegie Mellon University, Pittsburgh, PA
smitsch@cs.cmu.edu

Abstract. This paper describes a formal model of a "location, heading and speed" waypoint navigation task for an autonomous ground vehicle—that is, a task of navigating the vehicle towards a particular location so that it has the desired heading and speed when in that location. Our novel way of modeling this task makes formal reasoning over controller correctness tractable. We state our model in differential dynamic logic (dL), which we then use to establish a formal definition of waypoint feasibility and formally verify its validity in the KeYmaera X interactive theorem prover. The formal machine-checked proof witnesses that for any waypoint we consider feasible, the vehicle can indeed be controlled to reach it within the prescribed error bound. We also describe how we use these formal definitions and theorem statements to inform training of neural network controllers for performing this waypoint navigation task. Note that in our approach we do not need to rely on the neural network controller always being perfect—instead, the formal model allows a synthesis of a correct-by-construction safety net for the controller that checks whether the neural network output is safe to act upon and present a safe alternative if it is not.

1 Introduction

Our work is motivated by the task of assuring that an autonomous ground vehicle will safely travel from a start point to a destination point. Specifically, we are working with the US Army Combat Capability Development Center (CCDC) Ground Vehicle Systems Center (GVSC) autonomous Polaris MRZR vehicle, which was originally developed and used as part of the Dismounted Soldier Autonomy Tools (DSAT) effort [18]. At the highest level, we consider this "start to

* **Acknowledgment:** this material is based upon work supported by the United States Air Force and DARPA under Contract No. FA8750-18-C-0092. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the United States Air Force and DARPA. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

destination" task as a combination of two subtasks. The first subtask, which is outside the scope of this paper, is to plan a safe path from start to destination, replanning as necessary when circumstances change. The second subtask, addressed in this paper, is to then control the vehicle in a way that can be assured to follow the planned path without deviating from it beyond some error margin.

Obviously, not every path can be followed with sufficient precision—for example, a path that requires a turn that is much sharper than the vehicle's minimum turning radius could not be feasibly followed without significant deviation. Moreover, we are not only interested in limiting ourselves to paths that are feasible from the point of view of the vehicle's mechanical limits—we are interested in limiting ourselves to paths for which we can *assure* that the vehicle will be able to follow successfully. The notion of a *feasible path* effectively becomes the contract between the two subtasks—in assuring the overall task, we need to assure that the path planning subtask will always output a path (including the relevant error margins) that is both safe and feasible, and then assure that the path navigation subtask is capable of navigating any such feasible path without exceeding the error margin. Our choice of focusing on the latter task first is motivated by the desire to find a formal notion of feasible paths that is both sufficiently liberal (provide flexibility in the solution of the former task), while at the same time conservative enough to support our assurance argument.

The benefit of a formally verified notion of feasible paths is that it allows us to use machine learning to aggressively optimize a path following controller: The current "baseline" path following controller on the MRZR is a pure pursuit controller that does not always follow complex paths accurately, typically limited to fairly low speeds, and according to the users "feels somewhat robotic". The goal is to replace the MRZR path following controller with a neural network based one that is capable of driving both more aggressively, yet more accurately, and with better control. To provide safety guarantees for this critical task despite using machine learning, we need to be able to assure that the neural network based controller can be trusted to not misbehave—even in unlikely corner cases that might have never been encountered during training and testing.

This paper presents the first iteration of our work on assuring a path following controller, where we make a number of simplifying assumptions: *(i)* we use a point vehicle model with instantaneous steering,[3]; *(ii)* control is instantaneous and can change the vehicle actuation at any time, that is, our model of control is event-based (e.g., we can control a vehicle when it reaches a safety region boundary); and *(iii)* the vehicle is traversing even and flat terrain.

The goal is to first define a notion of feasible waypoint—those waypoints that we can assure the vehicle will be able to reach with some precision, given a particular starting state (or a particular set of possible starting states). Once we have a notion of a feasible waypoint, we can define a feasible path as a sequence of waypoints, such that the first waypoint in the sequence is feasible from the

[3] As a consequence of instantaneous steering, the curvature of the vehicle's path is able to change instantaneously to any value in the feasible range (that is, between $-\frac{1}{R_{\texttt{min}}}$ and $\frac{1}{R_{\texttt{min}}}$, where $R_{\texttt{min}}$ is the minimum turning radius of the vehicle).

vehicle starting state, and for each non-final waypoint, reaching that waypoint (within a specified error margin) implies that the subsequent waypoint is feasible.

In general, simply knowing the vehicle location is not enough to know whether the next waypoint will be feasible—at the very least, we need to know the vehicle heading and speed. Therefore, we specify waypoints in terms of: *(i)* desired location; *(ii)* desired heading; *(iii)* maximal speed at waypoint (should also not be exceeded prior to reaching the waypoint, unless it is unavoidable due to excessive initial speed); and *(iv)* error margin.

For the remainder of this paper, we focus on this specific subset of the general path following task—that of reaching a particular location, with desired heading at that location, and a speed limit at and prior to arrival at that location. We do not specifically restrict the route that the vehicle must take towards the desired location (it is the job of the route planner to place the waypoints close enough in those regions where the vehicle path needs to be tightly controlled), but prohibit the vehicle from driving "back"—that is, the angle between the vehicle's current heading and the desired heading at the waypoint must not exceed 180°.

Contributions. The main contributions of this work, summarized in Fig. 1, are the following. First, we develop a novel formalization of the "location, heading and speed" waypoint navigation task described above, expressing it in the differential dynamic logic (dL) modeling language [26,27] inside the KeYmaera X interactive theorem prover [13]. Second, we deduce and express the notions of "safe" states—those vehicle states where reaching the waypoint (subject to an error bound) is feasible—and "safe" control actions—namely, those control actions that would allow the vehicle to correctly navigate towards the waypoint. Third, we create a sequence of formal proofs in KeYmaera X establishing the desired properties—namely, as long as the waypoint is not yet reached, for each

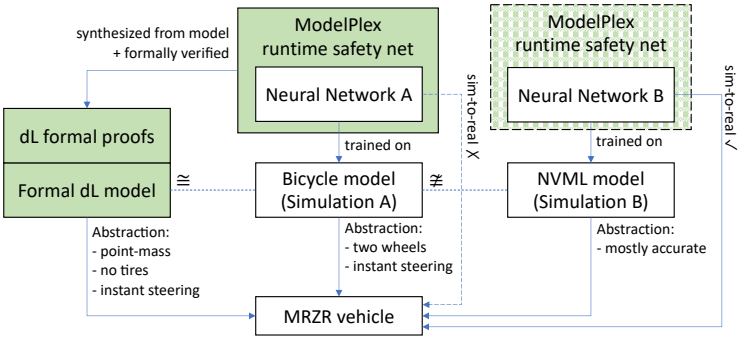


Fig. 1. Contributions overview: formal dL model, proofs, and runtime safety nets are formally verified; controllers are trained on simulations of varying fidelity, with varying sim-to-real success; verified safety net works well with the bicycle model simulation (fits closer to formal dL model), but additional work is needed to get to a formal model that closer matches the real vehicle.

safe state there is a safe control action and the use of any safe control actions implies to stay in the safe region until we reach the desired waypoint.

We want to emphasize that we use theorem proving not merely as a tool to obtain a correctness proof about an already correct system; we use it to explore and understand a system in all its subtleties and with all its corner cases thoroughly, to structure the system analysis process, to discover properties of the system that are not or only partially known (e.g., loop invariants), to discover and fix correctness bugs in the process, and to link model and system execution formally. The proof then ensures that the conditions we identified are sufficient. They are not necessarily the weakest necessary conditions—in fact, our initial formulation proved to be too conservative in practice and we subsequently revised the formalization and proofs to relax the safety condition to be reasonable in practice. The safety conditions we derived and proved correct can now be used as a basis for designing and ensuring correctness of other system parts.

It is also important to emphasize that "safe" in our case does not just mean "safe for now"; instead it means that we have proven that as long as the starting state is "safe", and only "safe" control actions are taken, the vehicle will remain "safe" until the waypoint is reached. Once the theorem is completed, following the usual KeYmaera X methodology, we use the ModelPlex tool [24] to automatically extract a correct-by-construction runtime monitoring safety net from the proof. The monitoring condition extracted by ModelPlex is a Boolean combination of a number of (in)equalities, and therefore the computational cost of evaluating the monitor on measurements and control output is negligible.

Our final contribution is the manner in which our formulation of the waypoint navigation task informed training neural networks to perform the task. Note that in our approach we do not rely on the neural network controller always being correct—instead, the correct-by-construction safety net for the controller checks whether the neural network output is safe to act upon, and presents a safe alternative if not (again, "safe" in a sense that it can ensure the vehicle will stay safe in the future, not just safe right now). Also note that while our focus is on neural networks, the same safety net could be used with any other controller.

Paper Structure. The remainder of this paper is structured as follows. First, in Section 2 we provide background information on differential dynamic logic (dL) and the KeYmaera X interactive theorem prover. Then, in Section 3 we describe the dL formulations of waypoint feasibility and navigation. Next, in Section 4 we outline the formal proof of waypoint feasibility, implemented in and machine-checked by KeYmaera X. In Section 5, we describe how we use the dL formulation of the task to inform training of a neural network controller for performing the task. Finally, in Section 6 we discuss related work and next steps.

2 Background

We express our models in differential dynamic logic (dL) [26,27]. dL is a specification and verification language with logical formulas that express properties of

Table 1. Hybrid programs

Statement	Meaning
$x := e$	Assigns value of term e to x
$x := *$	Assigns an arbitrary real value to x
$?Q$	Stays in current state if formula Q is true
$\{x' = f(x)$ & $Q\}$	Continuous evolution for any duration $t \geq 0$ with evolution domain constraint formula Q true throughout
$\alpha; \beta$	Executes β after α
$\alpha \cup \beta$	Executes either α or β, nondeterministically
α^*	Repeats α zero or more times

hybrid systems written in a programming language. Hybrid programs support differential equations (ODEs) as program statements, which allows us to express the control laws together with the entailed kinematics of the system and analyze control software for its correctness in terms of physical effects (e.g. collision avoidance). Proofs in differential dynamic logic are supported with the theorem prover KeYmaera X [13]. KeYmaera X is an LCF-style prover [22] with a small soundness-critical core of about 2000 LoC; the correctness of its core has been formally verified [4]. Table 1 summarizes the syntax and informal semantics of hybrid programs (detailed formal semantics are in [28]).

Typical control programs use assignments $x := e$ to compute intermediate or control output values. Random assignments $x := *$ choose any arbitrary real value for x and are often combined with tests $?Q$; for example, the program $x := *; ?0 \leq x < 5$ chooses any value in the half-open interval $[0, 5)$, the program $x := *; ?x^2 = y$ computes the square root of y. Tests $?Q$ control program execution: if the condition Q is true, program execution continues, otherwise it aborts (but may backtrack to another execution branch). Differential equations $x' = f(x)$ & Q follow a solution of $x' = f(x)$ for any duration as long as the constraint Q is true throughout. Note that even though the semantics is described in terms of solutions, in the proof calculus we do not rely on solving differential equations, but use invariance techniques for differential equations instead [30]. Sequential composition $\alpha; \beta$ first runs α and then β, non-deterministic choice $\alpha \cup \beta$ executes either α or β, and non-deterministic repetition α^* runs α any number of times (even zero). Other control instructions are expressible (e.g., if Q then α else β is expressed with $(?Q; \alpha) \cup (?\neg Q; \beta)$ with mutually exclusive conditions Q and $\neg Q$) and supported for convenience in the input syntax of KeYmaera X. It is good practice to avoid gaps in branch conditions (e.g., program $\big((?x \geq 5; x := 2) \cup (?x \leq 0; x := 2)\big)^*$ gets stuck after at most one repetition of the loop).

Many models use a controller that runs periodically, following the shape

$$(u := \mathrm{ctrl}(x); t := 0; \{x' = f(x, u), t' = 1 \ \& \ t \leq T\})^* \ ,$$

where a discrete program outputs control input u (slight abuse of notation: ctrl may output non-deterministically chosen values) for a subsequent continuous model $x' = f(x, u)$ with a clock $t' = 1$, repeated in a loop (*). The continuous

model runs for at most T time, as ensured by the time reset $t := 0$ and the constraint $t \leq T$, and then returns to $u := \mathrm{ctrl}(x)$. Note that the loop may repeat any number of times and so our safety analysis will hold for unbounded time. The differential equations in interesting continuous models often have non-polynomial solutions or are not solvable at all, but can still be analyzed in dL [30] with some careful rephrasing of transcendental functions. To make up for the lack of decidability of transcendental and trigonometric functions in the underlying real arithmetic theory[4], we encode trigonometric functions with differential equations as illustrated in the following example of a point moving along a unit circle. A typical model represents the position along the circle with angle θ and computes the position of the point in cartesian coordinates (x, y) using trigonometric functions sin and cos: the differential equation $x'=\cos\theta, y'=\sin\theta, \theta'=\omega$ describes change in position (x, y) and change in angle θ with angular velocity ω. The symbolic solution of this differential equation again mentions sin and cos and so results in undecidable arithmetic. To avoid this, we follow [29] to axiomatize sin and cos with differential equations using additional symbols $d_x=\cos\theta$ and $d_y=\sin\theta$. We determine the derivatives of d_x and d_y using

$$\begin{aligned} d_x' &= (\cos\theta)' = -\sin\theta\cdot\theta' = -d_y\cdot\omega \\ d_y' &= (\sin\theta)' = \quad \cos\theta\cdot\theta' = \quad d_x\cdot\omega \end{aligned}$$

and the resulting differential equation expands to $x'=d_x,\ y'=d_y,\ d_x'=-\omega d_y,\ d_y'=\omega d_x$, with the constraint $d_x^2+d_y^2=1$ for the starting values, which no longer uses trigonometric functions. To prove properties of such differential equations (including proving that the $d_x^2+d_y^2=1$ constraint is maintained), we use techniques in [30] instead of symbolic solutions.

We formalize properties with dL formulas per the following grammar where P, Q are formulas, $e, \tilde{e}$ are terms, x is a variable and α is a hybrid program:

$$P, Q ::= e \geq \tilde{e} \mid \neg P \mid P \wedge Q \mid P \vee Q \mid P \rightarrow Q \mid P \leftrightarrow Q \mid \forall x\, P \mid \exists x\, P \mid [\alpha]P$$

The operators of first-order real arithmetic are as usual with quantifiers ranging over the reals. For a hybrid program α and a dL formula P, $[\alpha]P$ is true *iff* P is true after all runs of α. It is particularly useful for expressing safety properties of the form $Q \rightarrow [\alpha]P$ with assumptions Q on the starting states of program α.

3 Modeling the Waypoint Navigation Task

Vehicle State. In a point-vehicle model, the state of a vehicle is defined by its position (x, y) and its velocity (v_x, v_y). We define the *left turning point* (l) and *right turning point* (r) as points that lie at equal distances from the center of the vehicle along a line transverse to the heading of the vehicle in left and right directions, respectively, and where the distance to the center is equal to the minimum radius of curvature ($R_{\mathtt{min}}$), as illustrated in Figure 2.

[4] Some solvers, e.g. dReal [14], opt for δ-decidability to render transcendental functions decidable.

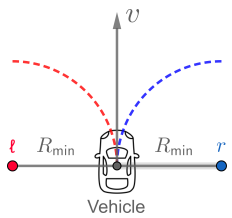


Fig. 2. Vehicle with left and right turning points.

Definition 1 (Vehicle State). *Coordinates of the left turning point l and right turning point r follow from vehicle position (x, y) and heading v, where (d_x, d_y) is a normalized direction of the vehicle:*

$$\begin{array}{llll} l_x = x - R_{min} d_y & r_x = x + R_{min} d_y & d_x = v_x / v & v = \sqrt{v_x^2 + v_y^2} \\ l_y = y + R_{min} d_x & r_y = y - R_{min} d_x & d_y = v_y / v & \end{array}$$

It turns out that it is significantly easier to use (l_x, l_y, r_x, r_y, v) as coordinates of the state than the traditional (x, y, v_x, v_y). In this coordinate system the waypoint feasibility constraints become simpler and much easier to both visualize and reason about. Therefore in our model a state of the vehicle is a point $state = (l_x, l_y, r_x, r_y, v)$ on a 4-dimensional manifold with condition:

$$\text{isWellformed}(state) ::= (l_x - r_x)^2 + (l_y - r_y)^2 = (2R_{\texttt{min}})^2 \ .$$

Vehicle Control and Movement. We define a control action for the vehicle as a pair $control = (a, \kappa)$, where a is the acceleration and κ is the curvature of the vehicle's path resulting from steering.

Definition 2 (Vehicle State Evolution). *The evolution of the vehicle state is specified by the following differential equations with control (a, κ):*

$$\begin{array}{lll} r_x' = v(l_y - r_y)(\mathrm{K} - \kappa)/2 & l_x' = v(l_y - r_y)(\mathrm{K} + \kappa)/2 & v' = a \\ r_y' = v(r_x - l_x)(\mathrm{K} - \kappa)/2 & l_y' = v(r_x - l_x)(\mathrm{K} + \kappa)/2 & \end{array}$$

Here $\mathrm{K} = \frac{1}{R_{min}}$ is the maximal curvature of a path the vehicle can take.

4 Proving Safety

4.1 Safe Starting States and Safe Actions

In Section 1 we outlined the notion of a *feasible waypoint*—that is a waypoint that, given a certain starting state, we can control the vehicle towards and can assure to reach within a given error bound. Here, we flip this relationship—rather than characterizing waypoints given a specific starting state, we will instead characterize starting states given a specific waypoint—namely a waypoint in the

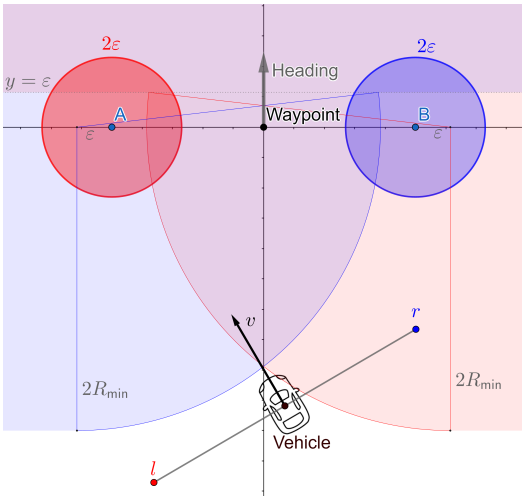


Fig. 3. Safety region for left and right turning centers of the vehicle starting state, when navigating to a waypoint at the origin. The state is considered to be safe when point l is outside of the light-red zone, point r is outside of the light-blue zone, and velocity v points upward. Dark red and dark blue circles are the destination zone: a vehicle is considered to have reached the waypoint when the point l is inside the dark-red circle and the point r inside the-dark blue circle.

origin $(0, 0)$ with the desired heading along the y axis facing towards positive values of y (this specific choice does not introduce a loss of generality as we can always perform a coordinate transformation to move and rotate the waypoint). We will call a state *safe* if the waypoint remains feasible with that starting state, and for a particular safe state, a safe action is one that does not immediately take us out of a safe state.

We say that we have reached the waypoint if the turning points l and r each reach the small neighborhood of points A and B respectively, where A and B are the points that lie distance $R_{\texttt{min}}$ from the desired waypoint to the left and right (see Fig. 3). Specifically, we require that each of $|l - A|$ and $|r - B|$ is $< 2\epsilon$ for some parameter ϵ, where $|\cdot|$ is Euclidean distance, and so a waypoint is considered reached by Def. 3.

Definition 3 (Waypoint reached). *For a waypoint at the origin, target region A for left turning point $l = (l_x, l_y)$ is at $(-R_{min}, 0)$ and target region B for right turning point $r = (r_x, r_y)$ is at $(R_{min}, 0)$, and so the waypoint is reached with tolerance ϵ when* Reached(*state*) *is satisfied:*

$$\text{Reached}(state) ::= (l_x + R_{min})^2 + l_y^2 < (2\epsilon)^2 \wedge (r_x - R_{min})^2 + r_y^2 < (2\epsilon)^2$$

Note, when $\epsilon \ll R_{\texttt{min}}$, then Reached(*state*) implies that both the vehicle location and the vehicle heading are close to the desired ones, while allowing some trade-off between the two.

The rest of this section proceeds as follows. First, we define a safe region as a certain subset of the state manifold. We also define a set of safe actions as a subset of the set of possible controls (a, κ). To establish that our definitions are valid, we then prove two theorems. First, we prove that whenever we are in a safe state, and we take a safe action for a period of time δ, we will remain in a safe state. Second, we prove whenever we are in a safe state, then unless we already reached our goal, there exists a safe action we can take.

4.2 Safe Region

We define our state safety condition as a conjunction of three conditions:

$$\text{Safe}(\mathit{state}) ::= \text{SafeDir}(\mathit{state}) \land \text{SafeLeft}(\mathit{state}) \land \text{SafeRight}(\mathit{state}) \text{ where}$$

- $\text{SafeDir}(\mathit{state}) ::= r_x \geq l_x$ captures the requirement that the vehicle is not allowed to drive "back",
- $\text{SafeLeft}(\mathit{state})$ is a proposition stating that the left turning point does not fall into the light red region shown in Fig. 3 and algebraically defined as a conjunction of the following clauses:
  - $\text{SafeFrontLineLeft}(\mathit{state}) ::= l_y < \epsilon$
  - $\text{SafeCircleLeft}(\mathit{state}) ::= l_y^2 + (l_x - R_{\texttt{min}} - \epsilon)^2 \geq (2R_{\texttt{min}})^2$
  - $\text{SafeBackLineLeft}(\mathit{state}) ::= l_y \leq -2R_{\texttt{min}} \lor l_x \leq R_{\texttt{min}} + \epsilon$
- $\text{SafeRight}(\mathit{state})$ is a proposition defined symmetrically to $\text{SafeLeft}(\mathit{state})$, stating that the right turning point does not fall into the light blue region shown in Fig. 3.

4.3 Safe Action

For each of the clauses $\text{SafeX}(\mathit{state})$ in the definition of safe region, we define a condition for a safe action $\text{SafeControlX}(\mathit{state}, \mathit{control})$ such that whenever the vehicle is in a state that satisfies property $\text{SafeX}(\mathit{state})$ and we take a control action that satisfies $\text{SafeControlX}(\mathit{state}, \mathit{control})$ for a period of time δ according to this control, then SafeX will be true for the vehicle's end state. That is,

$$\text{SafeX}(\mathit{state}) \land \text{SafeControlX}(\mathit{state}, \mathit{control}) \rightarrow [\text{Dynamics}]\, \text{SafeX}(\mathit{state})$$

where Dynamics is a set of differential equations of Def. 2 augmented with a time limit δ until the next control action: $t' = 1 \& t < \delta$. For example, we define

$$\begin{aligned}&\text{SafeControlBackLineLeft}(\mathit{state}, \mathit{control}) ::= \\ &\quad \big(l_y \leq -2R_{\texttt{min}} \land d_y(\mathrm{K}+\kappa)v\delta \leq -l_y\mathrm{K} - 2 - \Delta(\mathrm{K}+\kappa)v\delta\big) \\ &\quad \lor \left(l_x \leq R_{\texttt{min}} + \epsilon \land \begin{pmatrix} d_x(\mathrm{K}+\kappa)v\delta \leq -(l_x-\epsilon)\mathrm{K} + 1 - \Delta(\mathrm{K}+\kappa)v\delta \\ \lor\, d_x(\mathrm{K}+\kappa)v\delta \leq -(l_x-\epsilon)\mathrm{K} + 1 \land (l_x - r_x)\kappa \leq 0 \end{pmatrix}\right)\end{aligned}$$

where $\Delta = |v\kappa|\delta$ is the maximum angle the vehicle can turn with steering κ at velocity v during the reaction time δ. Then we prove that

$$\begin{aligned}&\text{SafeBackLineLeft}(\mathit{state}) \land \text{SafeControlBackLineLeft}(\mathit{state}, \mathit{control}) \rightarrow \\ &\qquad\qquad [\text{Dynamics}]\, \text{SafeBackLineLeft}(\mathit{state})\end{aligned}$$

The proofs of these lemmas are based on the following fairly straightforward fact that requires some care to prove in KeYmaera X. Suppose we have a point (x, y) that is moving on a unit circle $x^2 + y^2 = 1$ by the dynamic given by equations: $x' = -y; y' = x$. If at the starting point $x \geq a + \delta$, and $x \geq a$, $y < 0$, $1 - 2\delta \geq a$, then after time δ we will have that $x \geq a$. In case of $x \geq a + \delta$, this is true simply because $x' \geq -1$ and this case is easy to prove in KeYmaera X as well. For the case of $x \geq a$, $y < 0$, $1 - 2\delta \geq a$, this is true because the point is moving counterclockwise starting in the lower half of a circle and needs to move a distance of at least δ in order to cross the line $x = a$. This fact is key for establishing the various invariants, including those shown above, and was used many times in our proofs.

After proving such lemmas for all clauses we define SafeControl($state, control$) as a conjunction of these clauses.

Theorem 1. *Safe control keeps the vehicle in a safe state:*

$$\text{Safe}(state) \wedge \text{SafeControl}(state, control) \rightarrow [\text{Dynamics}]\, \text{Safe}(state)$$

Proof. By mechanized proof in KeYmaera X, splitting Safe into its conjuncts to apply lemmas. □

Theorem 2. *For every safe state, either the vehicle is at the destination, or there exists a control κ within the limits of maximum steering $K = \frac{1}{R_{min}}$ and a deadline D such that κ is safe up to deadline D:*

$$\begin{array}{l}\text{Safe}(state) \rightarrow \\ \quad \text{Reached}(state) \vee \\ \quad \exists |\kappa| \leq \text{K}.\, \exists D{>}0.\, \forall 0{<}\delta{\leq}D.\ \text{SafeControl}(state, control)\end{array}$$

Proof. By mechanized proof in KeYmaera X, along cases: a) not close to any of the bounds: go forward; b) close to a bound, then consider each of the following: go forward, or turn left, or turn right, or reached the waypoint, or unsatisfiable assumptions (case impossible). To formalize this proof in KeYmaera X, we used lemmas that allowed us to decompose statements under an existential quantifier:

$$\begin{array}{l}\big(\exists D_1{>}0.\, \forall 0{<}\delta{\leq}D_1.\, P(\delta)\big) \wedge \big(\exists D_2{>}0.\, \forall 0{<}\delta{\leq}D_2.\, Q(\delta)\big) \\ \quad \rightarrow \big(\exists D{>}0.\, \forall 0{<}\delta{\leq}D.\, P(\delta) \wedge Q(\delta)\big), \qquad \text{using } D = \min(D_1, D_2) \\ \big(\exists D{>}0.\, \forall 0{<}\delta{\leq}D.\, P(\delta)\big) \vee \big(\exists D{>}0.\, \forall 0{<}\delta{\leq}D.\, Q(\delta)\big) \\ \quad \rightarrow \big(\exists D{>}0.\, \forall 0{<}\delta{\leq}D.\, P(\delta) \vee Q(\delta)\big).\end{array}$$

These lemmas are used to prove statements of the form $\exists D{>}0.\, \forall 0{<}\delta{\leq}D.\, P(\delta)$ separately and construct the proofs by considering all cases. □

The two theorems together give us the desired safety property:

Corollary 1. *For each safe state there is a safe control action and we can use safe control actions to stay in the safe region, until we reach the desired waypoint.*

5 Training a Neural Network Controller

5.1 Controller Training

As we discussed in Section 4.1, we consider the waypoint reached, when points l and r that lie $R_{\texttt{min}}$ to the left and right of the vehicle (where "left" and "right" are taken with respect to the vehicle heading) each get within ϵ of the corresponding points A and B that lie $R_{\texttt{min}}$ to the left and right of the waypoint location (where "left" and "right" are taken with respect to the desired heading at the waypoint).

Reward Function. This definition informs a natural metric for how far the vehicle is from its goal—namely, $\sqrt{|l-A|^2+|r-B|^2}$ (where $|\cdot|$ is the Euclidean distance). Similarly, to Def. 3 (Reached), this naturally combines the location and heading errors in the same metric.

We use this distance metric as a reward function to train neural network controllers for waypoint navigation. At each time step, the reward is computed as $-\sqrt{|l-A|^2+|r-B|^2}-\lambda\max(0, v-v_{\texttt{max}})$, where λ is a positive meta-parameter constant. That is, at every time step the controller is penalized proportional to its distance to the target, and is additionally penalized whenever its speed exceeds the speed limit $v_{\texttt{max}}$.

Once the vehicle passes the $y = 0$ goal line, it is given a large reward if the waypoint is reached and a large penalty proportional to the distance to the waypoint—to encourage it to hit the waypoint as precisely as possible.

Training Parameters. We initially tried a number of algorithms for training the control NNs, including Trust Region Policy Optimization (TRPO) [32] and Constrained Policy Optimization (CPO) [1], but later settled on using a variation on the self-learned almost Lyapunov critics appoach [6], which is an extension of the Proximal Policy Optimization (PPO) algorithm [33]. We tried architectures with two and three dense hidden layers, and with hyperbolic tangent (tanh) and rectified linear unit (relu) activation functions, and eventually settled on using three hidden layers for the actor neural network, with 96, 32, and 16 nodes in each respectively, two hidden layers for the critic network, with 128 and 96 nodes in each respectively, relu activation for hidden layers, tanh activation for the output layer of the actor network, and softplus activation for the output layer of the critic network. In both cases, the chosen approach resulted in faster convergence as well as in faster and better-performing policies (as judged via a subjectively tuned combination of objective performance metrics).

Lessons Learned from an Earlier Prototype. In an early prototype, we only trained the controller on feasible waypoints. This worked well in simulation, but once we integrated it with the autonomy software stack [18] on the MRZR vehicle, it became obvious that an assured planner (the first subtask mentioned earlier) is crucial to ensure that the vehicle is never asked to navigate towards an infeasible waypoint. Without an assured planner, we must be prepared to make a best effort to also reach infeasible waypoints, even if there is no longer

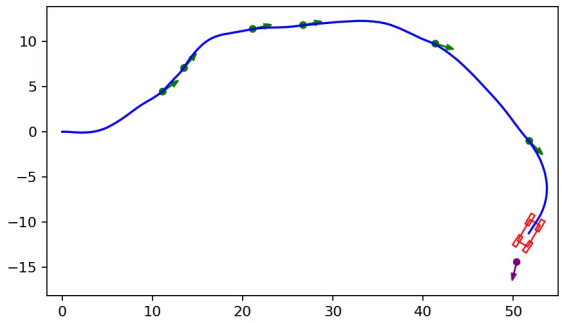


Fig. 4. Path of a simulated MRZR vehicle performing the waypoint+heading navigation task for a randomly chosen sequence of feasible waypoints, under an early neural network policy trained using the TRPO algorithm and only feasible waypoints, trained and tested in a simulator implementing a simple bicycle model of the vehicle, and a brush tire model of tire-to-surface interactions (Simulation A in Fig. 1).

any assurance of success. Further analysis revealed that in the early prototype the definition of feasible waypoint (or, equivalently, of safe starting state region) was overly conservative, so that too many of the waypoints generated by the (non-assured) planner were considered unsafe. As a consequence, we relaxed the notion of waypoint feasibility to the one presented in this paper, formally verified correctness of the relaxed definitions, and retrained the controllers.

5.2 Evaluating and Improving upon an Initial Controller

We initially trained and tested the neural network controller in a simulation that used a simple bicycle model of the vehicle, with a model of tire-to-surface interactions [2]. In that simulation environment, the neural network controller appeared to work very well, successfully reaching over 99% of the feasible waypoints chosen at random (Figure 4).

However, we soon discovered that the simulation used for initial training (Simulation A in Fig. 1) of the controller was insufficiently representative of the actual vehicle—e.g., the vehicle controls were more responsive and accurate in simulation than on the actual MRZR vehicle. The behavior of our early controller on the MRZR vehicle due to this mismatch can be seen in the video at `https://youtu.be/bE2UpKHxsLg`.

In order to account for the above challenges, we replaced the vehicle model we use for training the Neural Network with a model developed by CMU National Robotics Engineering Center (NREC) and implemented in their NREC Vehicle Modeling Library (NVML) based on the Wheeled Mobile Robot Dynamics Engine (WMRDE) [34,35], instantiated and calibrated to provide an accurate model of MRZR (Simulation B in Fig. 1). We then also incorporated a variant of the Neural Lyapunov technique [6] to improve controller stability and speed up training convergence.

The resulting neural network (NN) controller appeared to perform well in practice. Anecdotally, the GVSC team evaluating the work reported that: *(i)* NN sticks closer to the planned path than the baseline; *(ii)* NN navigates the path more consistently than the baseline; *(iii)* With NN, the planner does not get "stuck" replanning as often; *(iv)* Baseline feels robotic, while NN feels like a student driver; *(v)* Baseline is crawling, while NN goes more reliably fast, but NN accelerates too aggressively at times and does not feel sufficiently "in control" at high speeds; *(vi)* NN feels more natural—when going slowly, it feels like it could safely go faster; and *(vii)* NN usually drives smoother around obstacles, compared to the baseline.

5.3 Controller Safety

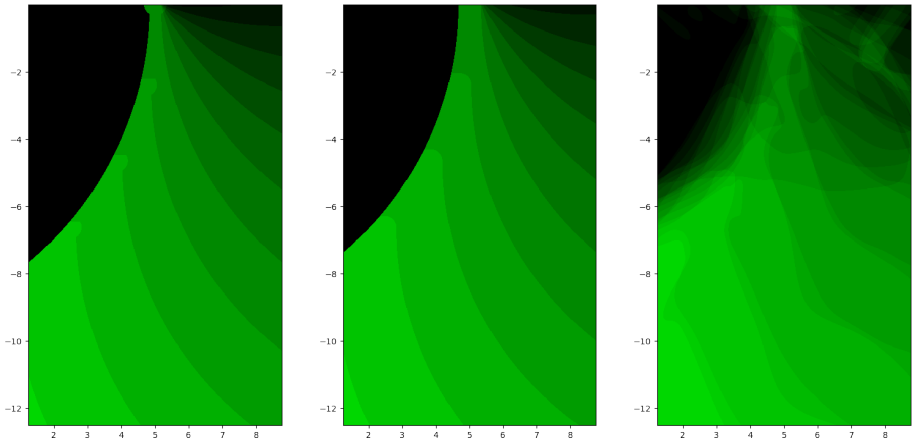
While our goal is to train a trustworthy controller that can be assured to always output a safe action (e.g., ether using the Neural Lyapunov based barrier function technique [7,6], or using the dReal SMT solver [14] to verify the properties of the neural network), neither is necessary in order to be able to assure safe vehicle behavior. Even if the underlying neural network based controller is not trustworthy, our technique allows for the use of the ModelPlex tool [24], which is part of the KeYmaera X theorem prover, to create a safe-by-construction safety net for the controller that checks whether the neural network outputs satisfy the SafeControl property and are therefore formally verified safe to act upon or else presents a safe alternative action (see Fig. 1).

However, our dL formalization in KeYmaera X used a simple vehicle model with instantaneous steering (similar to that used in Simulation A from Fig. 1). As we discussed above, the actual MRZR does not behave close enough to that model, and we ended up having to train the neural network controller with a significantly different model (Simulation B). As a result, we could use (and indeed successfully tested) the ModelPlex-created safety wrapper in Simulation A, but that safety wrapper is unfortunately not yet applicable to the real vehicle.

6 Discussion

6.1 Conclusions and Next Steps

Our initial attempts to formalize the task using the traditional (x, y, v_x, v_y) representation of the state resulted in large formulas that took hours to even try to simplify in Mathematica, and so it was fairly hopeless to try reasoning about them in a theorem prover. Our turn center based model not only simplifies the formalization, but makes it modular and easier to work with—namely, it allows decomposing the Safe(*state*) definition into three parts, each only referring to two of the coordinates—SafeDir only referring to r_x and l_x, SafeLeft only referring to l_x and l_y, and similarly SafeRight only referring to r_x and r_y. Such reduction in dimensionality is especially valuable given the doubly-exponential runtime complexity of deciding problems in real arithmetic [8]. This allowed us



(a) Feasible waypoints according to SafeRight as defined in Sect. 4.2

(b) Relaxed SafeRight[5]: areas of similar percentage are larger than original SafeRight

(c) Trained neural network critic's prediction of whether the $(0, 0)$ waypoint can be successfully reached

Fig. 5. SafeRight zone according to a verified formal model, a to-be-verified further relaxed formal model, and a neural network "critic"; $R_{\texttt{min}}$=5, ϵ≈0.17, vehicle speed 0.5 m/s, speed limit 0.5 m/s (deviation between simulator vehicle model for training the neural network and simplified formal model is smaller at low speed); color intensity of each point (r_x, r_y) indicates which percentage of orientations (i.e., starting configurations of fixed (r_x, r_y) and varying (l_x, l_y), so that the vehicle heading spans a range between -90°and 90°) would be safe; brighter is higher percentage.

to separately reason about different components of the state and different parts of the Safe constraint. Furthermore, it allowed us to exploit the left-right symmetry: once the "left" parts of the relevant lemmas are proved, tactics for proving the corresponding "right" lemmas follow immediately by symmetry.

The ability of the turn center based formalization to provide a natural "distance to waypoint" metric combining both the location and heading requirements, as well as expressing a natural trade-off between the two, results in a synergy between the formal modeling and verification of the "location + heading" waypoint navigation task and training of the corresponding neural network based controllers, where the distance metric informs the reward function. The synergy could be leveraged further, by explicitly penalizing the controller for choosing actions that our definitions consider unsafe. The result of such a constrained training process would be a controller that learns to navigate without causing monitor violations. For now, we want to test the conservativeness of our formal models and allow the training process to explore and discover its own notion of safety—and indeed, the learned controllers tend to converge to notions

[5] Relaxed SafeRight: circle of radius $2R_{\texttt{min}}-2\epsilon$ around destination point A excluded from safe zone and SafeFrontLineRight pushed back to 2ϵ.

of safety that are more aggressive than our formal models. In particular, our definition of SafeRight seems too conservative—it appears (although we have not been able to prove it just yet) that rather than excluding a circle of radius $2R_{\texttt{min}}$ around $(-R_{\texttt{min}} - \epsilon, 0)$ (light-blue quarter-circle in Fig. 3), it should be enough to only exclude a circle of radius $2R_{\texttt{min}} - \epsilon$, or perhaps even just $2R_{\texttt{min}} - 2\epsilon$ around the original point A (that is, $(-R_{\texttt{min}} - \epsilon, 0)$), and similarly it might be safe to push back SafeFrontLineRight to 2ϵ. The corresponding SafeRight zone and the SafeRight zone as discovered by the neural network "critic" (using the bicycle model of the vehicle) are illustrated in Fig. 5.

The KeYmaera X proof outlined in this paper took several man-months to complete and is to the best of our knowledge the largest safety proof ever done in KeYmaera X. It is at least 4x larger (in both the number of the interactive proof steps and the number of the primitive proof steps) than a previous "location + curvature" waypoint navigation safety proof performed by a different subset of our team [5], which was about the same size as other the largest KeYmaera X safety proofs. Despite that complexity, we are optimistic that we will be able to tweak it to support a less conservative Safe(*state*) definition with relatively little effort since KeYmaera X translates interactive proof steps into proof tactics. However what we ultimately hope to be able to do is to state and prove similar safety theorems using more realistic vehicle models, with multiple wheels, friction, actuation delays, and uneven terrain. At present, it seems unlikely that this would be feasible using our current approach of manually stating the definition, manually discovering relevant invariants, and manually transforming real arithmetic statements until they become tractable for the state-of-the-art real arithmetic solvers used by KeYmaera X. Instead, we plan to explore a machine-learning approach, where some aspects of the problem and some proof details are discovered and approximated by neural networks (along the lines of how the neural network "critic" in Figure 5 approximates our formal definition of SafeRight).

Another opportunity for future extension is to adapt the one-waypoint-at-a-time form of the navigation task to handle a sequence of multiple waypoints at once, which better resembles the current planner in the MRZR. While in our experience, the one-waypoint-at-a-time worked reasonably well (often better than the existing baseline implementation) even when used with a planner that generates waypoint sequences, we expect to be able to improve driving smoothness with a neural network controller that is provided with multiple waypoints.

6.2 Brief Comparison to Related Work

Formal verification of safety properties of robotic ground vehicles (see [21] for a comprehensive overview, or [11] for certification of autonomous systems) is roughly divided into methods based on reachability analysis and methods based on theorem proving: (online) reachability analysis [20,25] may support more complex differential equations at the expense of ignoring worst-case scenarios (e.g., [20] ignores the possibility of suddenly turning pedestrians), providing only

bounded-time results and requiring online computation, which makes it challenging to use for training with reinforcement learning with large volumes of simulations. Verisig [17] translates neural network controllers with sigmoid/tanh activation functions into a hybrid systems model, combined with a plant model, and uses bounded-time reachability analysis for verification of the resulting closed-loop system. NNV [37] provides several reachability algorithms a variety of set representations to analyze stability and bounded-time safety of neural networks.

In contrast, we obtain unbounded-time results offline and verified monitoring conditions [24] that are fast to evaluate even in reinforcement learning settings. Unbounded-time results for waypoint navigation are also obtained in [5] for a point-robot with a focus on integrated safety and liveness properties, but without guarantees on the orientation of the robot. Ways to introduce sensor and actuator disturbance in unbounded-time safety models are discussed in [23]. In [31] a planner for ground vehicle motion is verified in Isabelle, but path tracking and control is not addressed.

Complementary approaches use synthesis from formal specifications to obtain correct-by-construction control or planning, e.g., [3,40,41]. Unlike in our approach, the safety arguments of synthesis do not transfer to learned controllers. Approaches for monitoring and falsification based on formal specifications in linear temporal logic (LTL), metric temporal logic (MTL), or signal temporal logic (STL) target bug finding and runtime safety, treating monitor specifications as trusted input to the monitoring tool. For example [38] tests the control choices of autonomous vehicles with machine-learning components in the loop; the PLANrm framework [15] uses MTL to inspect and modify plans; [39] generate barrier certificates from simulation to justify safety of learned controllers; [10] use falsification to find when control obtained through machine learning violates specifications; [9] monitor the assumptions used for model-checking a control software; [19] uses runtime monitors for fault disambiguation on mobile robots. Several runtime verification tools provide convenient integration with robotic platforms, e.g., [16,36,12]. This gives confidence in safety *if* the monitor specification is correct and "enough" simulations and tests are conducted.

In our approach, safety is formally guaranteed and the verified models presented here are the source for ModelPlex monitor generation by proof [24]; ModelPlex guarantees mutually satisfied assumptions: the offline proof justifies monitor correctness, while satisfied online monitors justify model correctness and provably detect when the assumptions of the offline model are no longer true in reality. All artifacts in the offline proof and in the monitor synthesis proofs, including invariant regions of differential equations, barrier certificates, and other properties of differential equations if they come up in proofs, are justified from axioms in differential dynamic logic [27] and its differential equation axioms [30].

References

1. Achiam, J., Held, D., Tamar, A., Abbeel, P.: Constrained policy optimization. In: Proceedings of the 34th International Conference on Machine Learn-

ing, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. pp. 22–31 (2017), http://proceedings.mlr.press/v70/achiam17a.html
2. Ahn, E.: Towards Safe Reinforcement Learning in the Real World. Master's thesis, Carnegie Mellon University (Jul 2019), https://www.ri.cmu.edu/wp-content/uploads/2019/08/MSR_Thesis_-Edward_Ahn_2019.pdf, cMU-RI-TR-19-56
3. Alonso-Mora, J., DeCastro, J.A., Raman, V., Rus, D., Kress-Gazit, H.: Reactive mission and motion planning with deadlock resolution avoiding dynamic obstacles. Auton. Robots **42**(4), 801–824 (2018). DOI: 10.1007/s10514-017-9665-6
4. Bohrer, R., Rahli, V., Vukotic, I., Völp, M., Platzer, A.: Formally verified differential dynamic logic. In: Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs. ACM (Jan 2017). DOI: 10.1145/3018610.3018616
5. Bohrer, R., Tan, Y.K., Mitsch, S., Sogokon, A., Platzer, A.: A formal safety net for waypoint following in ground robots. IEEE Robotics and Automation Letters **4**(3), 2910–2917 (jul 2019). DOI: 10.1109/LRA.2019.2923099
6. Chang, Y.C., Gao, S.: Stabilizing neural control using self-learned almost Lyapunov critics. In: Proceedings of the 2021 International Conference on Robotics and Automation (ICRA 2021) (2021), https://arxiv.org/abs/2107.04989, to appear.
7. Chang, Y.C., Roohi, N., Gao, S.: Neural Lyapunov control (2020), https://arxiv.org/abs/2005.00611
8. Davenport, J.H., Heintz, J.: Real quantifier elimination is doubly exponential. J. Symb. Comput. **5**(1/2), 29–35 (1988). DOI: 10.1016/S0747-7171(88)80004-X
9. Desai, A., Saha, I., Yang, J., Qadeer, S., Seshia, S.A.: DRONA: a framework for safe distributed mobile robotics. In: Martínez, S., Tovar, E., Gill, C., Sinopoli, B. (eds.) Proceedings of the 8th International Conference on Cyber-Physical Systems, ICCPS 2017, Pittsburgh, Pennsylvania, USA, April 18-20, 2017. pp. 239–248. ACM (2017). DOI: 10.1145/3055004.3055022
10. Dreossi, T., Donzé, A., Seshia, S.A.: Compositional falsification of cyber-physical systems with machine learning components. J. Autom. Reasoning **63**(4), 1031–1053 (2019). DOI: 10.1007/s10817-018-09509-5
11. Fisher, M., Mascardi, V., Rozier, K.Y., Schlingloff, B., Winikoff, M., Yorke-Smith, N.: Towards a framework for certification of reliable autonomous systems. Auton. Agents Multi Agent Syst. **35**(1), 8 (2021). DOI: 10.1007/s10458-020-09487-2
12. Foughali, M., Bensalem, S., Combaz, J., Ingrand, F.: Runtime verification of timed properties in autonomous robots. In: 18th ACM/IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE 2020, Jaipur, India, December 2-4, 2020. pp. 1–12. IEEE (2020). DOI: 10.1109/MEMOCODE51338.2020.9315156
13. Fulton, N., Mitsch, S., Quesel, J., Völp, M., Platzer, A.: Keymaera X: an axiomatic tactical theorem prover for hybrid systems. In: Felty, A.P., Middeldorp, A. (eds.) Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9195, pp. 527–538. Springer (2015). DOI: 10.1007/978-3-319-21401-6_36
14. Gao, S., Kong, S., Clarke, E.M.: dreal: An SMT solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7898, pp. 208–214. Springer (2013). DOI: 10.1007/978-3-642-38574-2_14
15. Hoxha, B., Fainekos, G.E.: Planning in dynamic environments through temporal logic monitoring. In: Magazzeni, D., Sanner, S., Thiébaux, S. (eds.) Planning for

Hybrid Systems, Papers from the 2016 AAAI Workshop, Phoenix, Arizona, USA, February 13, 2016. AAAI Workshops, vol. WS-16-12. AAAI Press (2016), http://www.aaai.org/ocs/index.php/WS/AAAIW16/paper/view/12556
16. Huang, J., Erdogan, C., Zhang, Y., Moore, B.M., Luo, Q., Sundaresan, A., Rosu, G.: ROSRV: runtime verification for robots. In: Bonakdarpour, B., Smolka, S.A. (eds.) Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8734, pp. 247–254. Springer (2014). DOI: 10.1007/978-3-319-11164-3_20
17. Ivanov, R., Carpenter, T.J., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verifying the safety of autonomous systems with neural network controllers. ACM Trans. Embed. Comput. Syst. **20**(1), 7:1–7:26 (2021). DOI: 10.1145/3419742
18. Kania, R., Frederick, P., Pritchett, W., Wood, B., Mentzer, C., Johnson, E.: Dismounted soldier autonomy tools (DSAT) — from conception to deployment. In: 2014 NDIA Ground Vehicles Systems Engineering and Technology Symposium (2014), http://gvsets.ndia-mich.org/publication.php?documentID=171
19. Kempa, B., Zhang, P., Jones, P.H., Zambreno, J., Rozier, K.Y.: Embedding online runtime verification for fault disambiguation on robonaut2. In: Bertrand, N., Jansen, N. (eds.) Formal Modeling and Analysis of Timed Systems - 18th International Conference, FORMATS 2020, Vienna, Austria, September 1-3, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12288, pp. 196–214. Springer (2020). DOI: 10.1007/978-3-030-57628-8_12
20. Liu, S.B., Roehm, H., Heinzemann, C., Lütkebohle, I., Oehlerking, J., Althoff, M.: Provably safe motion of mobile robots in human environments. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017. pp. 1351–1357. IEEE (2017). DOI: 10.1109/IROS.2017.8202313
21. Luckcuck, M., Farrell, M., Dennis, L.A., Dixon, C., Fisher, M.: Formal specification and verification of autonomous robotic systems: A survey. ACM Comput. Surv. **52**(5), 100:1–100:41 (2019). DOI: 10.1145/3342355
22. Milner, R.: LCF: A way of doing proofs with a machine. In: Becvár, J. (ed.) Mathematical Foundations of Computer Science 1979, Proceedings, 8th Symposium, Olomouc, Czechoslovakia, September 3-7, 1979. Lecture Notes in Computer Science, vol. 74, pp. 146–159. Springer (1979). DOI: 10.1007/3-540-09526-8_11
23. Mitsch, S., Ghorbal, K., Vogelbacher, D., Platzer, A.: Formal verification of obstacle avoidance and navigation of ground robots. I. J. Robotics Res. **36**(12), 1312–1340 (2017). DOI: 10.1177/0278364917733549
24. Mitsch, S., Platzer, A.: ModelPlex: Verified runtime validation of verified cyber-physical system models. Form. Methods Syst. Des. **49**(1), 33–74 (2016). DOI: 10.1007/s10703-016-0241-z, special issue of selected papers from RV'14
25. Pan, Y., Lin, Q., Shah, H., Dolan, J.M.: Safe planning for self-driving via adaptive constrained ILQR. CoRR **abs/2003.02757** (2020), https://arxiv.org/abs/2003.02757
26. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reasoning **41**(2), 143–189 (2008). DOI: 10.1007/s10817-008-9103-8
27. Platzer, A.: A complete uniform substitution calculus for differential dynamic logic. J. Autom. Reasoning **59**(2), 219–265 (2017). DOI: 10.1007/s10817-016-9385-1
28. Platzer, A.: Logical Foundations of Cyber-Physical Systems. Springer (2018). DOI: 10.1007/978-3-319-63588-0
29. Platzer, A., Clarke, E.M.: Formal verification of curved flight collision avoidance maneuvers: A case study. In: Cavalcanti, A., Dams, D. (eds.) FM. LNCS, vol. 5850, pp. 547–562. Springer (2009). DOI: 10.1007/978-3-642-05089-3_35

30. Platzer, A., Tan, Y.K.: Differential equation invariance axiomatization. J. ACM **67**(1), 6:1–6:66 (2020). DOI: 10.1145/3380825
31. Rizaldi, A., Immler, F., Schürmann, B., Althoff, M.: A formally verified motion planner for autonomous vehicles. In: Lahiri, S.K., Wang, C. (eds.) Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11138, pp. 75–90. Springer (2018). DOI: 10.1007/978-3-030-01090-4_5
32. Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P.: Trust region policy optimization. CoRR **abs/1502.05477** (2015), http://arxiv.org/abs/1502.05477
33. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms (Aug 2017), http://arxiv.org/abs/1707.06347v2
34. Seegmiller, N.: Dynamic Model Formulation and Calibration for Wheeled Mobile Robots. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA (Oct 2014), https://www.ri.cmu.edu/publications/dynamic-model-formulation-and-calibration-for-wheeled-mobile-robots/
35. Seegmiller, N., Kelly, A.: High-fidelity yet fast dynamic models of wheeled mobile robots. IEEE Transactions on Robotics **32**(3), 614–625 (Jun 2016). DOI: 10.1109/TRO.2016.2546310
36. Shivakumar, S., Torfah, H., Desai, A., Seshia, S.A.: SOTER on ROS: A run-time assurance framework on the robot operating system. In: Deshmukh, J., Nickovic, D. (eds.) Runtime Verification - 20th International Conference, RV 2020, Los Angeles, CA, USA, October 6-9, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12399, pp. 184–194. Springer (2020). DOI: 10.1007/978-3-030-60508-7_10
37. Tran, H., Yang, X., Lopez, D.M., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12224, pp. 3–17. Springer (2020). DOI: 10.1007/978-3-030-53288-8_1
38. Tuncali, C.E., Fainekos, G., Prokhorov, D.V., Ito, H., Kapinski, J.: Requirements-driven test generation for autonomous vehicles with machine learning components. IEEE Trans. Intell. Veh. **5**(2), 265–280 (2020). DOI: 10.1109/TIV.2019.2955903
39. Tuncali, C.E., Kapinski, J., Ito, H., Deshmukh, J.V.: Reasoning about safety of learning-enabled components in autonomous cyber-physical systems. In: Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018. pp. 30:1–30:6. ACM (2018). DOI: 10.1145/3195970.3199852
40. Wong, K.W., Ehlers, R., Kress-Gazit, H.: Resilient, provably-correct, and high-level robot behaviors. IEEE Trans. Robotics **34**(4), 936–952 (2018). DOI: 10.1109/TRO.2018.2830353
41. Wong, K.W., Finucane, C., Kress-Gazit, H.: Provably-correct robot control with ltlmop, OMPL and ROS. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7, 2013. p. 2073. IEEE (2013). DOI: 10.1109/IROS.2013.6696636