International Journal of Robotics
36(12):1312–1340

DOI: 10.1177/0278364917733549
www.sagepub.com/
SAGE

Formal Verification of Obstacle Avoidance and Navigation of Ground Robots

Stefan Mitsch[1], Khalil Ghorbal[1,2], David Vogelbacher[1,3] and André Platzer[1]

Abstract

This article answers fundamental safety questions for ground robot navigation: Under which circumstances does which control decision make a ground robot safely avoid obstacles? Unsurprisingly, the answer depends on the exact formulation of the safety objective as well as the physical capabilities and limitations of the robot and the obstacles. Because uncertainties about the exact future behavior of a robot's environment make this a challenging problem, we formally verify corresponding controllers and provide rigorous safety proofs justifying why they can never collide with the obstacle in the respective physical model. To account for ground robots in which different physical phenomena are important, we analyze a series of increasingly strong properties of controllers for increasingly rich dynamics and identify the impact that the additional model parameters have on the required safety margins.

We analyze and formally verify: *(i) static safety*, which ensures that no collisions can happen with stationary obstacles, *(ii) passive safety*, which ensures that no collisions can happen with stationary or moving obstacles while the robot moves, *(iii)* the stronger *passive friendly safety* in which the robot further maintains sufficient maneuvering distance for obstacles to avoid collision as well, and *(iv) passive orientation safety*, which allows for imperfect sensor coverage of the robot, i. e., the robot is aware that not everything in its environment will be visible. We formally prove that safety can be guaranteed despite sensor uncertainty and actuator perturbation. We complement these provably correct safety properties with *liveness* properties: we prove that provably safe motion is flexible enough to let the robot navigate waypoints and pass intersections. In order to account for the mixed influence of discrete control decisions and the continuous physical motion of the ground robot, we develop corresponding *hybrid system* models and use *differential dynamic logic* theorem proving techniques to formally verify their correctness. Since these models identify a broad range of conditions under which control decisions are provably safe, our results apply to any control algorithm for ground robots with the same dynamics. As a demonstration, we, thus, also synthesize provably correct runtime monitor conditions that check the compliance of any control algorithm with the verified control decisions.

Keywords

provable correctness, obstacle avoidance, ground robot, navigation, hybrid systems, theorem proving

Introduction

Autonomous ground robots are increasingly promising as consumer products, ranging from today's autonomous household appliances Fiorini and Prassler (2000) to the driverless cars of the future being tested on public roads[a]. With the robots leaving the tight confounds of a lab or a locked-off industrial production site, robots face an increased need for ensuring safety, both for the sake of the consumer and the manufacturer. At the same time, less tightly structured environments outside a limited-access factory increase the flexibility and uncertainty of what other agents may do. This complicates the safety question, because it becomes even harder to achieve sufficiently exhaustive coverage of all possible behaviors.

Since the design of robot control algorithms is subject to many considerations and tradeoffs, the most useful safety results provide a broad characterization of the

[1] Computer Science Department, Carnegie Mellon University, Pittsburgh, USA
[2]Current address: INRIA, Rennes, France
[3]Current address: Karlsruhe Institute of Technology, Germany

Corresponding author:
Stefan Mitsch, Computer Science Department, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, USA
Email: smitsch@cs.cmu.edu

[a] http://www.nytimes.com/2010/10/10/science/10google.html?_r=0

set of control decisions that are safe in each of the states of the system. The control algorithms can then operate freely within the safe set of control decisions to optimize considerations such as reaching a goal or achieving secondary objectives without having to worry about safety. The resulting characterization of safe control actions serves as a "safety net" underneath any control algorithm, which isolates the safety question and provides strong safety guarantees for any ground robot following the respective physical dynamics.

One of the most important and challenging safety considerations in mobile robotics is to ensure that the robot does not collide with any obstacles Bouraine et al. (2012); Täubig et al. (2012); Wu and How (2012). Which control actions are safe under which circumstance crucially depends on the physical capabilities and limitations of the robot and moving obstacles in the environment. It also crucially depends on the exact formulation of the safety criterion, of which there are many for mobile robots Maček et al. (2009). We capture the former in a physical model describing the differential equations of continuous motion of a ground robot as well as a description of what discrete control actions can be chosen. This mix of discrete and continuous dynamics leads to a *hybrid system*. The safety criteria are formalized unambiguously in *differential dynamic logic* dℒ Platzer (2008, 2012a, 2017).

In order to justify the safety of the so-identified set of control decisions in the respective physical model, we formally verify the resulting controller and provide a rigorous proof in the dℒ theorem prover KeYmaera X Fulton et al. (2015). This proof provides undeniable mathematical evidence for the safety of the controllers, reducing safety of the robot to the question whether the appropriate physical model has been chosen for the robot and its environment. Due to the uncertainties in the exact behavior of the robot and the agents in its environment, a range of phenomena are important in the models.

We consider a series of models with static obstacles at fixed positions, dynamic obstacles moving with bounded velocities, sensors with limited field of vision, sensor uncertainties, and actuator disturbances. We identify the influence of each of those on the required design of safe controllers. We also consider a series of safety criteria that account for the specific features of these models, since one of the subtle conceptual difficulties is what safety even means for an autonomous robot. We would want it to be always collision-free, but that requires other vehicles to be reasonable, e. g., not actively try to run into our robot when it is just stopped in a corner. One way of doing that is to assume stringent constraints on the behavior of obstacles Loos et al. (2011); Bouraine et al. (2012).

In this article, we refrain from doing so and allow obstacles with an arbitrary continuous motion respecting a known upper bound on their velocity. Then our robot is safe, intuitively, if no collision can ever happen *where the robot is to blame*. For static obstacles, the situation is easy, because the robot is to blame for every collision that happens, so our safety property and its proof show that the robot will never collide with any static obstacle (*static safety*). For dynamic obstacles, safety is subtle, because other moving agents might actively try to ruin safety and cause collisions even if our robot did all it could to prevent them. We analyze *passive safety* Maček et al. (2009), which requires that the robot does not actively collide, i. e., collisions only happen when a moving obstacle ran into the robot while the robot was stopped. Our proofs guarantee passive safety with minimal assumptions about obstacles. The trouble with passive safety is that it still allows the robot to stop in unsafe places, creating unavoidable collision situations in which an obstacle has no control choices left that would prevent a collision. *Passive friendly safety* Maček et al. (2009) addresses this challenge with more careful robot decisions that respect the dynamic limitations of moving obstacles (e. g., their braking capabilities). A passive-friendly robot not only ensures that it is itself able to stop before a collision occurs, but it also maintains sufficient maneuvering room for obstacles to avoid a collision as well. Finally, we introduce *passive orientation safety*, which restricts the responsibility of the robot to avoid collisions to only parts of the robot's surroundings (e. g., the robot is responsible for collisions with obstacles to its front and sides, but obstacles are responsible when hitting the robot from behind). We complement these safety notions with liveness proofs to show that our provably safe controllers are flexible enough to let the robot navigate waypoints and cross intersections.

All our models use symbolic bounds so our proofs hold *for all* choices of the bounds. As a result, we can account for uncertainty in several places (e. g., by instantiating upper bounds on acceleration or time with values including uncertainty). We show how further uncertainty that cannot be attributed to such bounds (in particular location uncertainty, velocity uncertainty, and actuator uncertainty) can be modeled and verified *explicitly*.

The class of control algorithms we consider is inspired by the *dynamic window* algorithm Fox et al. (1997), but is equally significant for other control algorithms when combining our results of provable safety with verified runtime validation Mitsch and Platzer (2016). Unlike related work on obstacle avoidance (e. g., Althoff et al. (2012); Pan et al. (2012); Täubig et al. (2012); Seward et al. (2007); van den Berg et al. (2011)), we use *hybrid system* models and verification techniques that describe and verify the robot's discrete control choices *along with its continuous, physical motion*.

In summary, our contributions are *(i)* hybrid system models of navigation and obstacle avoidance control

algorithms of robots, *(ii)* safety proofs that guarantee static safety, passive safety, passive friendly safety, and passive orientation safety in the presence of stationary and moving obstacles despite sensor uncertainty and actuator perturbation, and *(iii)* liveness proofs that the safety measures are flexible enough to allow the robot to reach a goal position and pass intersections. The models and proofs of this article are available[b] in the theorem prover KeYmaera X Fulton et al. (2015) unless otherwise noted. They are also cross-verified with our previous prover KeYmaera Platzer and Quesel (2008). This article extends our previous safety analyses Mitsch et al. (2013) with orientation safety for less conservative driving, as well as with liveness proofs to guarantee progress. In order to take the vagaries of the physical environment into account, these guarantees are for hybrid system models that include discrete control decisions, reaction delays, differential equations for the robot's physical motion, bounded sensor uncertainty, and bounded actuator perturbation.

Related Work

Isabelle has recently been used to formally verify that a C program implements the specification of the dynamic window algorithm Täubig et al. (2012). We complement such effort by formally verifying the correctness of the dynamic window algorithm while considering continuous physical motion.

PASSAVOID Bouraine et al. (2012) is a navigation scheme designed to operate in unknown environments by stopping the robot before it collides with obstacles (passive safety). The validation was however only based on simulations. In this work, we provide formal guarantees while proving the stronger passive friendly safety ensuring that the robot does not create unavoidable collision situations by stopping in unsafe places.

Wu and How (2012) assume unpredictable behavior for obstacles with known forward speed and maximum turn rate. The robot's motion is however explicitly excluded from their work which differs from the models we prove.

We generalize the safety verification of straight line motions Loos et al. (2011); Mitsch et al. (2012) and the two-dimensional planar motion with constant velocity Loos et al. (2013a); Platzer and Clarke (2009) by allowing translational and rotational accelerations.

Pan et al. (2012) proposes a method to smooth the trajectories produced by sampling-based planners in a collision-free manner. Our article proves that such trajectories are indeed safe when considering the control choices of a robot and its continuous dynamics.

LQG-MP van den Berg et al. (2011) is a motion planning approach that takes into account the sensors, controllers, and motion dynamics of a robot while working with uncertain information about the environment. The approach attempts to select the path that decreases the collision probability. Althoff et al. (2012) use a probabilistic approach to rank trajectories according to their collision probability. They propose a collision cost metric to refine the ranking based on the relative speeds and masses of the collision objects. Seward et al. (2007) try to avoid potentially hazardous situations by using Partially Observable Markov Decision Processes. Their focus, however, is on a user-definable trade-off between safety and progress towards a goal. Safety is not guaranteed under all circumstances. We rather focus on formally proving collision-free motions under reasonable assumptions of the environment.

It is worth noting that formal methods were also used for other purposes in the hybrid systems context. For instance, in Plaku et al. (2009, 2013), the authors combine model checking and motion planning to efficiently falsify a given property. Such lightweight techniques could be used to increase the trust in the model but are not designed to prove the property. LTLMoP Sarid et al. (2012) enables the user to specify high-level behaviors (e. g., visit all rooms) when the environment is continuously updated. The approach synthesizes plans, expressed in linear temporal logic, of a hybrid controller, whenever new map information is discovered while preserving the state and task completion history of the desired behavior. In a similar vein, the automated synthesis of controllers restricted to straight-line motion and satisfying a given property formalized in linear temporal logic has been recently explored in Kress-Gazit et al. (2009), and adapted to discrete-time dynamical systems in Wolff et al. (2014). Karaman and Frazzoli (2012) explore optimal trajectory synthesis from specifications in deterministic μ-calculus.

Preliminaries: Differential Dynamic Logic

A robot and the moving obstacles in its environment form a *hybrid system*: they make discrete control choices (e. g., compute the actuator set values for acceleration, braking, or steering), which in turn influence their actual physical behavior (e. g., slow down to a stop, move along a curve). In test-driven approaches, simulators or field tests provide insight into the expected physical effects of the control code. In formal verification, hybrid systems provide joint models for both discrete and continuous behavior, since verification of either component alone does not capture the full behavior of a robot and its environment. In this section, we first give an overview of the relationship between testing, simulation, and formal verification, before we introduce the syntax and semantics of the specification language that we use for formal verification.

[b] http://web.keymaeraX.org/show/ijrr/robix.kyx

Testing, Simulation, and Formal Verification

Testing, simulation, and formal verification complement each other. Testing helps to make a system robust under real-world conditions, whereas simulation lets us execute a large number of tests in an inexpensive manner (at the expense of a loss of realism). Both, however, show correctness for the finitely many tested scenarios only. Testing and simulation discover the presence of bugs, but cannot show their absence. Formal verification, in contrast, provides precise and undeniable guarantees for *all* possible executions of the modeled behavior. Formal verification either discovers bugs if present, or shows the absence of bugs in the model, but, just like simulation, cannot show whether or not the model is realistic. In Section Monitoring for Compliance At Runtime, we will see how we can use runtime monitoring to bridge both worlds. Testing, simulation, and formal verification all base on similar ingredients, but apply different levels of rigor as follows.

Software. Testing and simulation run a specific control algorithm with specific parameters (e. g., run a specific version of an obstacle avoidance algorithm with maximum velocity $V = 2\,\text{m/s}$). Formal verification can specify symbolic parameters and nondeterministic inputs and effects and, thereby, capture entire families of algorithms and many scenarios at once (e. g., verify all velocities $0 \leq v \leq V$ for any maximum velocity $V \geq 0$ at once).

Hardware and physics. Testing runs a real robot in a real environment. Both simulation and formal verification, in contrast, work with *models* of the hardware and physics to provide sensor values and compute how software decisions result in real-world effects.

Requirements. Testing and simulation can work with informal or semi-formal requirements (e. g., a robot should not collide with obstacles, which leaves open the question whether a slow bump is considered a collision or not). Formal verification uses mathematically precise formal requirements expressed as a logical formula (without any ambiguity in their interpretation distinguishing precisely between correct behavior and faults).

Process. In testing and simulation, requirements are formulated as test conditions and expected test outcomes. A test procedure then runs the robot several times under the test conditions and one manually compares the actual output with the expected outcome (e. g., run the robot in different spaces, with different obstacles, various software parameters, and different sensor configurations to see whether or not any of the runs fail to avoid obstacles). The test protocol serves as correctness evidence and needs to be repeated when anything changes. In formal verification, the requirements are formulated as a logical formula. A theorem prover then creates a mathematical proof showing that *all* possible executions—usually infinitely many—of the model are correct (safety proof), or showing that the model has a way to achieve a goal (liveness proof). The mathematical proof is the correctness certificate.

Differential Dynamic Logic

This section briefly explains the language that we use for formal verification. It explains *hybrid programs*, which is a program notation for describing hybrid systems, and *differential dynamic logic* dL Platzer (2008, 2010a, 2012a, 2017), which is the logic for specifying and verifying correctness properties of hybrid programs. Hybrid programs can specify how a robot and obstacles in the environment make decisions and move physically. With differential dynamic logic we specify formally which behavior of a hybrid program is considered correct. dL allows us to make statements that we want to be true for all runs of a hybrid program (safety) or for at least one run (liveness).

One of the many challenges of developing robots is that we do not know the behavior of the environment exactly. For example, a moving obstacle may or may not slow down when our robot approaches it. In addition to programming constructs familiar from other languages (e. g., assignments and conditional statements), hybrid programs, therefore, provide nondeterministic operators that allow us to describe such unknown behavior of the environment concisely. These nondeterministic operators are also useful to describe parts of the behavior of our own robot (e. g., we may not be interested in the exact value delivered by a position sensor, but only that it is within some error range), which then corresponds to verifying an entire family of controllers at once. Using nondeterminism to model our own robot has the benefit that later optimization (e. g., mount a better sensor or implement a faster algorithm) does not necessarily require re-verification since variations are already covered.

Table 1 summarizes the syntax of hybrid programs together with their informal semantics. Many of the operators will be familiar from regular expressions, but the discrete and continuous operators are crucial to describe robots. A common and useful assumption when working with hybrid systems is that time only passes in differential equations, but discrete actions do not consume time (whenever they do consume time, it is easy to transform the model to reflect this just by adding explicit extra delays).

We now briefly describe each operator with an example. Assignment $x := \theta$ instantaneously assigns the value of the term θ to the variable x (e. g., let the robot choose maximum braking). Nondeterministic assignment $x := *$ assigns an arbitrary real value to x (e. g., an obstacle may choose any acceleration, we do not know which value exactly). Sequential composition $\alpha; \beta$ says that β starts after α finishes (e. g., $a := 3;\ r := *$ first let the robot choose acceleration to be 3, then choose any steering angle).

Table 1. Hybrid program representations of hybrid systems.

Statement	Effect
$x := \theta$	assign current value of term θ to variable x (discrete assignment)
$x := *$	assign arbitrary real number to variable x
$\alpha;\ \beta$	sequential composition, first run α, then β
$\alpha \cup \beta$	nondeterministic choice, follow either α or β
α^*	nondeterministic repetition repeats α any $n \geq 0$ number of times
$?F$	check that a condition F holds in the current state, and abort run if it does not
$(x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, Q)$	evolve x_i along differential equation system $x_i' = \theta_i$ for any amount of time restricted to maximum evolution domain Q

The nondeterministic choice $\alpha \cup \beta$ follows either α or β (e. g., the obstacle may slow down or speed up). The nondeterministic repetition operator α^* repeats α zero or more times (e. g., the robot may encounter obstacles over and over again, or wants to switch between the options of a nondeterministic choice, but we do not know exactly how often). The continuous evolution $x' = \theta \,\&\, Q$ evolves x along the differential equation $x' = \theta$ for any arbitrary amount of time within the evolution domain Q (e. g., the velocity of the robot decreases along $v' = -b \,\&\, v \geq 0$ according to the applied brakes $-b$, but does not become negative since hitting the brakes won't make the robot drive backwards). The test $?F$ checks that the formula F holds, and aborts the run if it does not (e. g., test whether the distance to an obstacle is large enough to continue driving). Other nondeterministic choices may still be possible if one run fails, which explains why an execution of hybrid programs with backtracking is a good intuition.

A typical pattern with nondeterministic assignment and tests is to limit the assignment of arbitrary values to known bounds (e. g., limit an arbitrarily chosen acceleration to the physical limits of the robot, as in $a := *;\ ?(a \leq A)$, which says a is any value less or equal A). Another useful pattern is a nondeterministic choice with complementary tests $(?P;\alpha) \cup (?\neg P;\beta)$, which models an if-then-else statement if (P) α else β.

The dL formulas can be formed according to the following grammar (where $\sim$ is any comparison operator in $\{<, \leq, =, \geq, >, \neq\}$ and θ_1, θ_2 are arithmetic expressions in $+, -, \cdot, /$ over the reals):

$$\phi ::= \theta_1 \sim \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \forall x\, \phi \mid [\alpha]\phi \mid \langle\alpha\rangle\phi$$

Further operators, such as Euclidean norm $\|\theta\|$ and infinity norm $\|\theta\|_\infty$ of a vector θ, are definable from these. The formula $[\alpha]\phi$ is true in a state if and only if *all* runs of hybrid program α from that state lead to states in which formula ϕ is true. The formula $\langle\alpha\rangle\phi$ is true in a state if and only if there is at least one run of hybrid program α to a state in which formula ϕ is true.

In particular, dL formulas of the form $F \rightarrow [\alpha]G$ mean that if F is true in the initial state, then all executions of the hybrid program α only lead to states in which formula G is true. Dually, formula $F \rightarrow \langle\alpha\rangle G$ expresses that if F is true in the initial state then there is a state reachable by the hybrid program α that satisfies formula G.

Proofs in Differential Dynamic Logic

Differential dynamic logic comes with a verification technique to prove correctness properties Platzer (2008, 2010a, 2012a, 2017). The underlying principle behind a proof in dL is to symbolically decompose a large hybrid program into smaller and smaller pieces until the remaining formulas no longer contain the actual programs, but only their logical effect. For example, the effect of a simple assignment $x := 1 + 1$ in a proof of formula $[x := 1 + 1]x = 2$ results in the proof obligation $1 + 1 = 2$. The effects of more complex programs may of course not be as obviously true. Still, whether or not these remaining formulas in real arithmetic are valid is decidable by a procedure called *quantifier elimination* Collins (1975).

Proofs in dL consist of three main aspects: *(i)* find invariants for loops and differential equations, *(ii)* symbolically execute programs to determine their effect, and finally *(iii)* verify the resulting real arithmetic with external solvers for quantifier elimination. High modeling fidelity becomes expensive in the arithmetic parts of the proof, since real arithmetic is decidable but of high complexity Davenport and Heintz (1988). As a result, proofs of high-fidelity models may require arithmetic simplifications (e.g., reduce the number of variables by abbreviating complicated terms, or by hiding irrelevant facts) before calling external solvers.

The reasoning steps in a dL proof are justified by dL axioms. The equivalence axiom $[\alpha \cup \beta]\phi \leftrightarrow [\alpha]\phi \wedge [\beta]\phi$, for example, allows us to prove safety about a program with a nondeterministic choice $\alpha \cup \beta$ by instead proving safety of the program α in $[\alpha]\phi$ and separately proving safety of the program β in $[\beta]\phi$. Reducing all occurrences of $[\alpha \cup \beta]\phi$ to corresponding conjunctions $[\alpha]\phi \wedge [\beta]\phi$, which are handled separately, successively decomposes safety questions for a hybrid program of the form $\alpha \cup \beta$ into safety questions for simpler subsystems.

The theorem prover KeYmaera X Fulton et al. (2015) implements a uniform substitution proof calculus for dL Platzer (2017) that checks all soundness-critical side conditions during a proof. KeYmaera X also provides significant automation by bundling axioms into larger *tactics* that perform multiple reasoning steps at once. For example, when proving safety of a program with a loop

$A \to [\alpha^*]S$, a tactic for loop induction tries to find a loop invariant J to split the proof into three separate, smaller pieces: one branch to show that the invariant is true in the beginning ($A \to J$), one branch to show that running the program α without loop once preserves the invariant ($J \to [\alpha]J$), and another branch to show that the invariant is strong enough to guarantee safety ($J \to S$). If an invariant J cannot be found automatically, users can still provide their own guess or knowledge about J as input to the tactic. Differential invariants provide a similar inductive reasoning principle for safety proofs about differential equations ($A \to [x' = \theta]S$) without requiring symbolic solutions, so they can be used to prove properties about non-linear differential equations, such as for robots. Differential invariants can be synthesized for certain classes of differential equations Sogokon et al. (2016).

The tactic language Fulton et al. (2017) of KeYmaera X can also be used by users for scripting proofs to provide human guidance when necessary. We performed all proofs in this paper in the verification tool KeYmaera X Fulton et al. (2015) and/or its predecessor KeYmaera Platzer and Quesel (2008). While all our proofs ship with KeYmaera, we provide all but one proof also in its successor KeYmaera X, which provides rigorous verification from a small soundness-critical core, comes with high-assurance correctness guarantees from cross-verification results Bohrer et al. (2017) in the theorem provers Isabelle and Coq, and enables us to provide succinct tactics that produce the proofs and facilitate easier reuse of our verification results. Along with the fact that KeYmaera X supports hybrid systems with nonlinear discrete jumps and nonlinear differential equations, these advantages make KeYmaera X more readily applicable to robotic verification than other hybrid system verification tools. SpaceEx Frehse et al. (2011), for example, focuses on (piecewise) linear systems. KeYmaera X implements automatic proof strategies that decompose hybrid systems symbolically. This compositional verification principle helps scaling up verification, because KeYmaera X verifies a big system by verifying properties of subsystems. Strong theoretical properties, including relative completeness, have been shown for dℒ Platzer (2008, 2012b, 2017).

Preliminaries: Obstacle Avoidance with the Dynamic Window Approach

The robotics community has come up with an impressive variety of robot designs, which differ not only in their tool equipment, but also (and more importantly for the discussion in this article) in their kinematic capabilities. This article focuses on wheel-based ground vehicles. In order to make our models applicable to a large variety of robots, we use only limited control options (e. g., do not move sideways to avoid collisions since Ackermann drive could not follow such evasion maneuvers). We consider robots that drive forward (non-negative translational velocity) in sequences of arcs in two-dimensional space. If the radius of such a circle is large, the robot drives (forward) on an approximately straight line. Such trajectories can be realized by robots with single-wheel drive, differential drive (wheels may rotate in opposite directions), Ackermann drive (front wheels steer), synchro-drive (all wheels steer), or omni-directional drive (wheels rotate in any direction) Bräunl (2006). In a nutshell, *in order to stay on the safe side, our models conservatively underestimate the capabilities of our robot while conservatively overestimating the dynamic capabilities of obstacles*.

Many different navigation and obstacle avoidance algorithms have been proposed for such robots, e. g. *dynamic window* Fox et al. (1997), *potential fields* Khatib (1985), or *velocity obstacles* Fiorini and Shiller (1998). For an introduction to various navigation approaches for mobile robots, see Bonin-Font et al. (2008); Choset et al. (2005). The inspiration for the algorithm we consider in this article is the dynamic window algorithm Fox et al. (1997), which is derived from the motion dynamics of the robot and thus discusses all aspects of a hybrid system (models of discrete and continuous dynamics). But other control algorithms including path planners based on RRT LaValle and Kuffner (2001) or A* Hart et al. (1968) are compatible with our results when their control decisions are checked with a runtime verification approach Mitsch and Platzer (2016) against the safety conditions we identify for the motion here.

The dynamic window algorithm is an obstacle avoidance approach for mobile robots equipped with synchro drive Fox et al. (1997) but can be used for other drives too Brock and Khatib (1999). It uses circular trajectories that are uniquely determined by a translational velocity v together with a rotational velocity ω, see Section Robot and Obstacle Motion Model below for further details. The algorithm is organized into two steps: *(i)* The range of all possible pairs of translational and rotational velocities is reduced to admissible ones that result in safe trajectories (i. e., avoid collisions since those trajectories allow the robot to stop before it reaches the nearest obstacle) as follows (Fox et al. 1997, (14)): $V_a = \{(v,\omega) \mid v \leq \sqrt{2\text{dist}(v,\omega)v_b'} \land \omega \leq \sqrt{2\text{dist}(v,\omega)\omega_b'}\}$ This definition of admissible velocities, however, neglects the reaction time of the robot. Our proofs reveal the additional safety margin that is entailed by the reaction time needed to revise decisions. The admissible pairs are further restricted to those that can be realized by the robot within a short time frame t (the dynamic window) from current velocities v_a and ω_a to account for acceleration effects despite assuming velocity to be a

piecewise constant function in time (Fox et al. 1997, (15)): V_d=$\{(v,\omega) \mid v \in [v_a - v't, v_a + v't] \wedge \omega \in [\omega_a - \omega' t, \omega_a + \omega' t]\}$. Our models, instead, control acceleration and describe the effect on velocity in differential equations. If the set of admissible and realizable velocities is empty, the algorithm stays on the previous safe curve (such curve exists unless the robot started in an unsafe state). *(ii)* Progress towards the goal is optimized by maximizing a goal function among the set of all admissible controls. For safety verification, we can omit step *(ii)* and verify the stronger property that *all* choices fed into the optimization are safe. Even if none is identified, the previous safe curve can still be continued.

Robot and Obstacle Motion Model

This section introduces the robot and obstacle motion models that we are using throughout the article. Table 2 summarizes the model variables and parameters of both the robot and the obstacle for easy reference. In the following subsections, we illustrate their meaning in detail.

Robot State and Motion

The dynamic window algorithm safely abstracts the robot's shape to a single point by increasing the (virtual) shapes of all obstacles correspondingly (cf. Minguez et al. (2006) for an approach to attribute robot shape to obstacles). We also use this abstraction to reduce the verification complexity. Fig. 1 illustrates how we model the position p, orientation d, and trajectory of a robot.

The robot has state variables describing its current position $p = (p_x, p_y)$, translational velocity $s \geq 0$, translational acceleration a, orientation vector[c] $d = (\cos\theta, \sin\theta)$, and angular velocity[d] $\theta' = \omega$. The translational and rotational velocities are linked w.r.t. the rigid body planar motion by

Table 2. Parameters, state variables of robot and obstacle

	2D	Description
p	(p_x, p_y)	Position of the robot
s		Translational speed
a		Translational acceleration, s.t. $-b \leq a \leq A$
ω		Rotational velocity, s.t. $\omega r = s$
d	(d_x, d_y)	Orientation of the robot, s.t. $\|d\| = 1$
c	(c_x, c_y)	Curve center, s.t. $d = (p - c)^{\perp}$
r		Curve radius, s.t. $r = \|p - c\|$
o	(o_x, o_y)	Position of the obstacle
v	(v_x, v_y)	Translational velocity, including orientation, s.t. $\|v\| \leq V$
A		Maximum acceleration $A \geq 0$
b		Minimum braking $b > 0$
ε		Maximum control loop reaction delay $\varepsilon > 0$
V		Maximum obstacle velocity $V \geq 0$
Ω		Maximum rotational velocity $\Omega \geq 0$

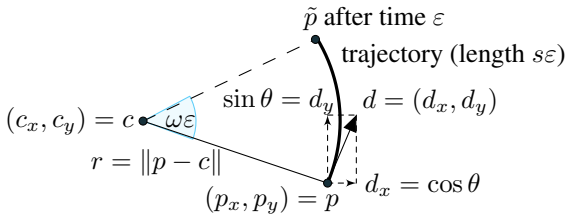


Figure 1. State illustration of a robot on a two-dimensional plane. The robot has position $p = (p_x, p_y)$, orientation $d = (d_x, d_y)$, and drives on circular arcs (thick arc) of radius r with translational velocity s, rotational velocity ω and thus angle $\omega\varepsilon$ around curve center points $c = (c_x, c_y)$. In time ε the robot will reach a new position $\tilde{p}$, which is $s\varepsilon$ away from the initial position p when measured along the robot's trajectory arc.

the formula $r\omega = s$, where the curve radius $r = \|p - c\|$ is the distance between the robot and the center of its current curve $c = (c_x, c_y)$. The usual modeling approach with angle θ and trigonometric functions $\sin\theta$ and $\cos\theta$ to determine the position along a curve, however, results in undecidable arithmetic. Instead, we encode sine and cosine functions in the dynamics using the extra variables $d_x = \cos\theta$ and $d_y = \sin\theta$ by *differential axiomatization* Platzer (2010b). The continuous dynamics for the dynamic window algorithm Fox et al. (1997) can, thus, be described by the differential equation system of ideal-world dynamics of the planar rigid body motion:

$$p' = sd, s' = a, d' = \omega d^{\perp}, (r\omega)' = a$$

where

- $p' = sd$ represents $p'_x = sd_x, p'_y = sd_y$ in vectorial notation,
- the condition $d' = \omega d^{\perp}$ is vector notation for the rotational dynamics $d'_x = -\omega d_y, d'_y = \omega d_x$ where $^{\perp}$ is the orthogonal complement, and
- the condition $(r\omega)' = a$ encodes the rigid body planar motion $r\omega = s$ that we consider.

The dynamic window algorithm assumes piecewise constant velocity s between decisions despite accelerating, which is physically unrealistic. We, instead, control acceleration a and do not perform instant changes of the velocity. Our model is closer to the actual dynamics of a robot. The realizable velocities follow from the differential equation system according to the controlled acceleration a.

We assume bounds for the permissible acceleration a in terms of a maximum acceleration $A \geq 0$ and braking power

[c] As stated earlier, we study unidirectional motion: the robot moves along its direction, that is the vector d gives the direction of the velocity vector.
[d] The derivative with respect to time is denoted by prime ($'$).

$b > 0$, as well as a bound Ω on the permissible rotational velocity ω. We use ε to denote the upper bound for the control loop time interval (e. g., sensor and actuator delays, sampling rate, and computation time). That is, the robot might react quickly, but it can take no longer than time ε to react. The robot would not be safe without such a time bound, because its control might then never run. In our model, all these bounds will be used as symbolic parameters and not concrete numbers. Therefore, our results apply to *all values* of these parameters and can be enlarged to include uncertainty.

Obstacle State and Motion

An obstacle has (vectorial) state variables describing its current position $o = (o_x, o_y)$ and velocity $v = (v_x, v_y)$. The obstacle model is deliberately liberal to account for many different obstacle behaviors. The only restriction about the dynamics is that the obstacle moves continuously with bounded velocity $\|v\| \leq V$ while the physical system evolves for ε time units. The original dynamic window algorithm considers the special case of $V = 0$ (obstacles are stationary). Depending on the relation of V to ε, moving obstacles can make quite a difference, e. g., when other fast robots or the soccer ball meet slow communication-based virtual sensors as in RoboCup.[e]

Safety Verification of Ground Robot Motion

We want to prove motion safety of a robot whose controller tries to avoid obstacles. Starting from a simplified robot controller, we develop increasingly more realistic models, and discuss different safety notions. *Static safety* describes a vehicle that never collides with stationary obstacles. *Passive safety* Maček et al. (2009) considers a vehicle to be safe if no collisions happen while it moves (i. e., the vehicle does not itself collide with obstacles, so if a collision occurs at all then while the vehicle was stopped). The intuition is that if collisions happen while our robot is stopped, then it must be the moving obstacle's fault. Passive safety, however, puts some of the burden of avoiding collisions on other objects. We, thus, also prove the stronger *passive friendly safety* Maček et al. (2009), which guarantees that our robot will come to a stop safely under all circumstances and will leave sufficient maneuvering room for moving obstacles to avoid a collision.[f] Finally, we prove *passive orientation safety*, which accounts for limited sensor coverage of the robot and its orientation to reduce the responsibility of the robot in structured spaces, such as on roads with lanes.

Table 3 gives an overview of the safety notions (both formally and informally) and the assumptions made about the robot and the obstacle in our models. We consider all four models and safety properties to show the differences between the required assumptions and the safety guarantees that can be made. The verification effort and complexity difference is quite instructive. Static safety provides a strong guarantee with a simple safety proof, because only the robot moves. Passive safety can be guaranteed by proving safety of all robot choices, whereas passive friendly safety requires additional liveness proofs for the obstacle. In the following sections, we discuss models and verification of the collision avoidance algorithm in detail.

For the sake of clarity, we initially make the following simplifying assumptions to get an easier first model:

- A1 in its decisions, the robot will use maximum braking or maximum acceleration, no intermediate controls,
- A2 the robot will not reverse its direction, but only drive smooth curves in forward direction, and
- A3 the robot will not keep track of the center of the circle around which its current trajectory arc is taking it, but chooses steering through picking a curve radius.

In Section Refined Models for Safety Verification we will see how to remove these simplifications again.

The subsections are structured as follows: we first discuss the rationale behind the model (see paragraphs *Modeling*) and provide an intuition why the control choices in this model are safe (see paragraphs *Identification of Safe Controls*). Finally, we formally verify the correctness of the model, i. e., use the model in a correctness theorem and summarize the proof that the control choices indeed guarantee the model to satisfy the safety condition (see paragraphs *Verification*). Whether the model adequately represents reality is a complementary question that we discuss in Section Monitoring for Compliance At Runtime.

Static Safety with Maximum Acceleration

In environments with only stationary obstacles, static safety ensures that the robot will never collide.

Modeling The prerequisite for obtaining a formal safety result is to first formalize the system model in addition to its desired safety property. We develop a model of the collision avoidance algorithm as a hybrid program, and express static safety as a safety property in dℒ.

As in the dynamic window algorithm, the collision avoidance controller uses the distance to the nearest obstacle for every possible curve to determine admissible velocities (e. g., compute distances in a loop and pick the obstacle with the smallest). Instead of modeling the algorithm for searching the nearest obstacle and computing

[e] http://www.robocup.org/

[f] The robot ensures that there is enough room for the obstacle to stop before a collision occurs. If the obstacle decides not to, then the obstacle is to blame and our robot is still considered safe.

Table 3. Overview of safety notions, responsibilities of the robot and its assumptions about obstacles

Safety	Responsibility of Robot	Assumptions about Obstacles
Static (Model 2)	Positive distance to all stationary obstacles $\|p-o\| > 0$	Obstacles remain stationary and never move $v = 0$
	Safety (cf. Theorem 1, feasible initial conditions ϕ_{ss}): $\phi_{ss} \rightarrow [\text{Model 2}](\|p-o\| > 0)$	
Passive (Model 3)	Positive distance to all obstacles while driving $s \neq 0 \rightarrow \|p-o\| > 0$	Known maximum velocity V of obstacles $0 \leq v \leq V$
	Safety (cf. Theorem 2, feasible initial conditions ϕ_{ps}): $\phi_{ps} \rightarrow [\text{Model 3}](s \neq 0 \rightarrow \|p-o\| > 0)$	
Passive Friendly (Model 4+5)	Sufficient maneuvering space for obstacles $s \neq 0 \rightarrow \|p-o\| > \frac{V^2}{2b_o} + \tau V$	Known maximum velocity V, minimum braking capability b_o, and maximum reaction time τ $0 \leq v \leq V \land b_o > 0 \land \tau \geq 0$
	Safety (cf. Theorem 3, feasible initial conditions ϕ_{pfs}): robot retains space $\phi_{pfs} \rightarrow [\text{Model 4}](s \neq 0 \rightarrow \|p-o\| > \frac{V^2}{2b_o} + \tau V)$ obstacles can avoid collision $\phi_{pfs} \land s = 0 \land \|p-o\| > \frac{V^2}{2b_o} + \tau V \rightarrow \langle \text{Model 5} \rangle (\|p-o\| > 0 \land v = 0)$	
Passive Orientation (Model 6)	Positive distance to all obstacles while driving, unless an invisible obstacle interfered with the robot while the robot cautiously stayed inside its observable region $s \neq 0 \rightarrow (\|p-o\| > 0 \lor (\mathit{isVisible} \leq 0 \land \|\beta\| < \gamma))$	Known maximum velocity V of obstacles $0 \leq v \leq V$
	Safety (cf. Theorem 4): $\phi_{pos} \rightarrow [\text{Model 6}](s \neq 0 \rightarrow \|p-o\| > 0 \lor (\mathit{isVisible} \leq 0 \land \|\beta\| < \gamma))$	

its closest perimeter point explicitly, our model exploits the power of nondeterminism to model this concisely. It nondeterministically picks *any* obstacle $o := (*, *)$ and tests its safety. Since the choice of the obstacle to consider was nondeterministic and the model is only safe if it is safe for *all* possible ways of selecting *any* obstacle nondeterministically, this includes safety for the closest perimeter point of the nearest obstacle (ties are included) and is thus safe for all possible obstacles. Explicit representations of multiple obstacles will be considered in Section Arbitrary Number of Obstacles.

In the case of non-point obstacles, o denotes the obstacle perimeter point that is closest to the robot (this fits naturally to obstacle point clouds delivered by radar and Lidar sensors, from which the closest point on the arc will be chosen). In each controller run of the robot, the position o is updated nondeterministically (to consider any obstacle including the ones that now became closest). In this process, the robot may or may not discover a new safe trajectory. If it does, the robot can follow that new safe trajectory w.r.t. any nondeterministically chosen obstacle. If not, the robot can still brake on the previous trajectory, which was shown to be safe in the previous control cycle for any obstacle, including the obstacle chosen in the current control cycle.

Model 1 summarizes the robot controller, which is parameterized with a *drive* action and a condition *safe* identifying when it is safe to take this *drive* action. The formula *safe* is responsible for selecting control choices that keep the robot safe when executing control action *drive*.

Model 1 Parametric robot controller model

$$\begin{aligned} &\mathit{ctrl}_r(\boldsymbol{\mathit{drive}}, \boldsymbol{\mathit{safe}}) \equiv \\ &\quad (a := -b) && (1)\\ &\quad \cup\, (?(s = 0);\ a := 0;\ \omega := 0) && (2)\\ &\quad \cup\, (\boldsymbol{\mathit{drive}};\ \omega := *;\ ?(-\Omega \leq \omega \leq \Omega); && (3)\\ &\qquad r := *;\ o := (*, *);\ ?(\mathit{curve} \land \boldsymbol{\mathit{safe}})) && (4)\\ &\mathit{curve} \equiv r \neq 0 \land r\omega = s && (5) \end{aligned}$$

The robot is allowed to brake at all times since the assignment that assigns full braking to a in (1) has no test. If the robot is stopped ($s = 0$), it may choose to stay in its current spot without turning, cf. (2). Finally, if it is safe to accelerate, which is what formula parameter *safe* determines, then the robot may choose a new safe curve in its dynamic window. That is, it performs action *drive* (e.g. maximum acceleration) and chooses any rotational velocity in the bounds, cf. (3) and computes the corresponding radius r according to the condition (5). This corresponds to testing all possible rotational velocity values at the same time and choosing some that passes condition *safe*. An implementation in an imperative language would use loops to enumerate all possible values and all obstacles and test

each pair (s, ω) separately w.r.t. every obstacle, storing the admissible pairs in a data structure (as e. g., in Täubig et al. (2012)).

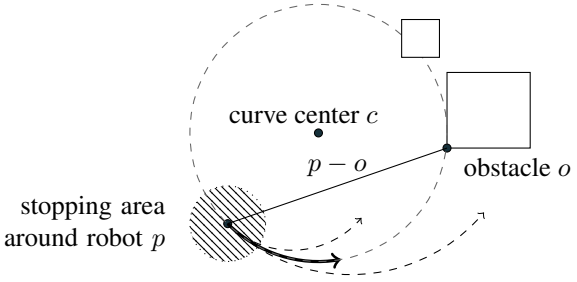


Figure 2. Illustration of static safety: the robot must stop before reaching the closest obstacle on a curve (three of infinitely many curves illustrated). Obstacles with shapes reduce to single points by considering the perimeter point that is closest to the robot.

The curve is determined by the robot following a circular trajectory of radius r with angular velocity ω starting in initial direction d, cf. (4). The trajectory starts at p with translational velocity s and rotational velocity ω, as defined by $r\omega = s$ in (5). This condition ensures that we simultaneously pick an admissible angular velocity ω according to (Fox et al. 1997, (14)) when choosing an admissible velocity s. Together with the orientation d of the robot, which is tangential to the curve, this also implicitly characterizes the rotation center c; see Fig. 2. We will explicitly represent the rotation center in Appendix Passive Safety for Sharp Turns for more aggressive maneuvering. For starters, we only need to know how to steer by r. For the sake of clarity we restrict the study to circular trajectories with non-zero radius ($r \neq 0$ so that the robot is not spinning on the spot). We do not include perfectly straight lines in our model, but instead mimic the control principle of a real robot that will control periodically to adjust for actuator perturbation and drift when trying to drive straight lines, so it suffices to approximate straight-line driving with large curve radii (r approaches infinity). The sign of the radius signifies if the robot follows the curve in clockwise ($r < 0$) or counter-clockwise direction ($r > 0$). Since $r \neq 0$, the condition $(r\omega)' = a$ can be rewritten as differential equation $\omega' = \frac{a}{r}$. The distance to the nearest obstacle on that curve is measured by $o := (*, *)$ in (4).

Model 2 represents the common controller-plant model: it repeatedly executes the robot control choices followed by dynamics, cf. (6). Recall that the arbitrary number of repetitions is indicated by the * at the end. The continuous dynamics of the robot from Section Robot and Obstacle Motion Model above is defined in (8)–(10) of Model 2.

Identification of Safe Controls The most critical element of Model 2 is the choice of the formula $safe_{ss}$ in (7) that we chose for parameter *safe*. This formula is responsible for selecting control choices that keep the robot safe. While its ultimate justification will be the safety proof (Theorem 1), this section explains intuitively why we chose the particular design in (7). Generating such conditions is possible, see Quesel et al. (2016) for an approach how to phrase conjectures with unknown constraints in dL and use theorem proving to discover constraints that make a formula provable.

Model 2 Dynamic window with static safety

$$dw_{ss} \equiv \left(ctrl_r(a := A\ ,\ safe_{ss}); dyn_{ss}\right)^* \tag{6}$$

$$safe_{ss} \equiv \|p - o\|_\infty > \frac{s^2}{2b} + \left(\frac{A}{b} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon s\right) \tag{7}$$

$$dyn_{ss} \equiv t := 0;\ \{t' = 1,\ p' = sd,\ s' = a, \tag{8}$$

$$d' = \omega d^\perp,\ \omega' = \frac{a}{r} \tag{9}$$

$$\&\ s \geq 0 \wedge t \leq \varepsilon\} \tag{10}$$

A circular trajectory of radius r ensures static safety if it allows the robot to stop before it collides with the nearest obstacle. Consider the extreme case where the radius $r = \infty$ is infinitely large and the robot, thus, travels on a straight line. In this case, the distance between the robot's current position p and the nearest obstacle o must account for the following components: First, the robot needs to be able to brake from its current velocity s to a complete stop (equivalent to (Fox et al. 1997, (14)) characterizing admissible velocities), which takes time $\frac{s}{b}$ and requires distance $\frac{s^2}{2b}$:

$$\frac{s^2}{2b} = \int_0^{s/b} (s - bt)dt\ . \tag{11}$$

Second, it may take up to ε time until the robot can take the next control decision. Thus, we must take into account the distance that the robot may travel w.r.t. the maximum acceleration A and the distance needed for compensating its acceleration of A during that reaction time with braking power b (compensating for the speed increase $A\varepsilon$ takes time $\frac{A\varepsilon}{b}$):

$$\left(\tfrac{A}{b} + 1\right)\left(\tfrac{A}{2}\varepsilon^2 + \varepsilon s\right) = \int_0^{\varepsilon} (s + At)dt + \int_0^{A\varepsilon/b} (s + A\varepsilon - bt)dt\ . \tag{12}$$

The safety distance chosen for $safe_{ss}$ in (7) of Model 2 is the sum of the distances (11) and (12). The safety proof will have to show that this construction was indeed safe and

that it is also safe for all other curved trajectories that the obstacle and robot could be taking in the model instead.

To simplify the proof's arithmetic, we measure the distance between the robot's position p and the obstacle's position o in the infinity-norm $\|p-o\|_\infty$, i. e., either $|p_x-o_x|$ or $|p_y-o_y|$ must be safe. In the illustrations, this corresponds to replacing the circles representing reachable areas with outer squares. This over-approximates the Euclidean norm distance $\|p-o\|_2 = \sqrt{(p_x-o_x)^2+(p_y-o_y)^2}$ by a factor of at most $\sqrt{2}$.

Verification With the proof calculus of dL Platzer (2008, 2012a, 2010a, 2017), we verify the safety of the control algorithm in Model 2. The robot is safe, if it maintains positive distance $\|p-o\|>0$ to (nondeterministic so any) obstacle o (see Table 3), i. e., it always satisfies:

$$\psi_{\text{ss}} \;\equiv\; \|p-o\| > 0 \;. \tag{13}$$

In order to guarantee ψ_{ss} always holds, the robot must stay at a safe distance, which still allows the robot to brake to a complete stop before hitting any obstacle. The following condition captures this requirement as an invariant φ_{ss} that we prove to hold for all executions of the loop in (6):

$$\varphi_{\text{ss}} \;\equiv\; \|p-o\| > \frac{s^2}{2b} \;. \tag{14}$$

Formula (14) says that the robot and the obstacle are safely apart. In this case, the safe distance in the loop invariant coincides with (11), which describes the stopping distance.

We prove that the property (13) holds for all executions of Model 2 (so also all obstacles) under the assumption that we start in a state satisfying the symbolic parameter assumptions ($A\geq 0$, $V\geq 0$, $\Omega\geq 0$, $b>0$, and $\varepsilon>0$) as well as the following initial conditions:

$$\phi_{\text{ss}} \;\equiv\; s=0 \wedge \|p-o\|>0 \wedge r\neq 0 \wedge \|d\|=1 \;. \tag{15}$$

The first two conditions of the conjunction formalize that the robot is stopped at a safe distance initially. The third conjunct states that the robot is not spinning initially. The last conjunct $\|d\|=1$ says that the direction d is a unit vector. Any other formula ϕ_{ss} implying invariant φ_{ss} is a safe starting condition as well (e. g., driving with sufficient space, so invariant φ_{ss} itself).

Theorem 1. Static safety. *Robots following Model 2 never collide with stationary obstacles as expressed by the provable* dL *formula* $\phi_{ss} \rightarrow [dw_{ss}]\psi_{ss}$.

Proof. We proved Theorem 1 for circular trajectories in KeYmaera X. The proof uses the invariant φ_{ss} (14) for handling the loop. It uses *differential cuts* with *differential invariants* (16)–(20)—an induction principle for differential equations Platzer (2012c)—to prove properties about *dyn* without requiring symbolic solutions.

$$t \geq 0 \tag{16}$$
$$\|d\| = 1 \tag{17}$$
$$s = \text{old}(s) + at \tag{18}$$
$$-t\left(s-\frac{a}{2}t\right) \leq p_x - \text{old}(p_x) \leq t\left(s-\frac{a}{2}t\right) \tag{19}$$
$$-t\left(s-\frac{a}{2}t\right) \leq p_y - \text{old}(p_y) \leq t\left(s-\frac{a}{2}t\right) \tag{20}$$

The differential invariants capture that time progresses (16), that the orientation stays a unit vector (17), that the new speed s is determined by the previous speed $\text{old}(s)$ and the acceleration a (18) for time t, and that the robot does not leave the bounding square of half side length $t(s-\frac{a}{2}t)$ around its previous position $\text{old}(p)$ (19)–(20). The function $\text{old}(\cdot)$ is shorthand notation for an auxiliary or ghost variable that is initialized to the value of $\cdot$ before the ODE. □

Passive Safety with Maximum Acceleration

In the presence of moving obstacles, collision freedom gets significantly more involved, because, even if our robot is doing the best it can, other obstacles could still actively try to crash into it. Passive safety, thus, considers the robot safe if no collisions can happen while it is driving. The robot, thus, needs to be able to come to a full stop before making contact with any obstacle, see Fig. 3.

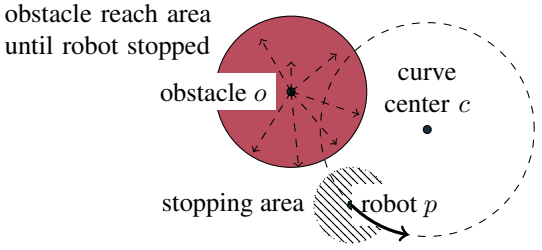


Figure 3. Illustration of passive safety: the area reachable by the robot until it can stop must not overlap with the area reachable by the obstacle during that time.

Intuitively, when every moving robot and obstacle follows passive safety then there will be no collisions. Otherwise, if careless or malicious obstacles are moving in the environment, passive safety ensures that at least our own robot is stopped so that collision impact is kept small. In this section, we will develop a robot controller that provably ensures passive safety. We remove the restriction that obstacles cannot move, but the robot and the obstacle will decide on their next maneuver at the same time and they are still subject to the simplifying assumptions A1–A3.

Modeling We refine the collision avoidance controller and model to include moving obstacles, and state its passive

Model 3 Dynamic window with passive safety

$$dw_{\text{ps}} \equiv (ctrl_o; ctrl_r(a := A\ ,\ safe_{\text{ps}}); dyn_{\text{ps}})^* \tag{21}$$

$$\boldsymbol{ctrl_o \equiv v := (*,*);\ ?\|v\| \leq V} \tag{22}$$

$$safe_{\text{ps}} \equiv \|p-o\|_\infty > \frac{s^2}{2b} + \boldsymbol{V}\frac{\boldsymbol{s}}{\boldsymbol{b}} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(s+\boldsymbol{V})\right) \tag{23}$$

$$dyn_{\text{ps}} \equiv t := 0;\ \{t' = 1,\ \boldsymbol{p_o'} = \boldsymbol{v},\ p_r' = sd,\ v_r' = a,\ d_r' = \omega d^\perp,\ \omega_r' = \frac{a}{r}\ \&\ s \geq 0 \wedge t \leq \varepsilon\} \tag{24}$$

safety property in dℒ. In the presence of moving obstacles all obstacles must be considered and tested for safety. The main intuition here is that all obstacles will respect a maximum velocity V, so the robot is safe when it is safe for the worst-case behavior of the nearest obstacle. Our model again exploits the power of nondeterminism to model this concisely by picking *any* obstacle $o := (*,*)$ and testing its safety. In each controller run of the robot, the position o is updated nondeterministically (which includes the ones that are now closest because the robot and obstacles moved). If the robot finds a new safe trajectory, then it will follow it (the velocity bound V ensures that all obstacles will stay more distant than the worst-case of the nearest one chosen nondeterministically). Otherwise, the robot will stop on the current trajectory, which was tested to be safe in the previous controller decision.

Model 3 follows a setup similar to Model 2. The continuous dynamics of the robot and the obstacle as presented in Section Robot and Obstacle Motion Model above are defined in (24) of Model 3.

The control of the robot is executed after the control of the obstacle, cf. (21). Both robot and obstacle only write to variables that are read in the dynamics, but not in the controller of the respective other agent. Therefore, we could swap the controllers to $ctrl_r; ctrl_o$, or use a nondeterministic choice of $(ctrl_o; ctrl_r) \cup (ctrl_r; ctrl_o)$ to model independent parallel execution Müller et al. (2016). Fixing one specific ordering $ctrl_o; ctrl_r$ reduces proof effort, because it avoids branching the proof into all the different possible execution orders (which in this case differ only in their intermediate computations but have the same effect on motion).

The obstacle may choose any velocity in any direction up to the maximum velocity V assumed about obstacles ($\|v\| \leq V$), cf. (22). This uses the modeling pattern from Section Preliminaries: Differential Dynamic Logic. We assign an arbitrary (two-dimensional) value to the obstacle's velocity ($v := (*,*)$), which is then restricted by the maximum velocity with a subsequent test ($?\|v\| \leq V$). Overall, (22) allows obstacles to choose an arbitrary velocity in any direction, but at most of speed V. Analyzing worst-case situations with a powerful obstacle that supports sudden direction and velocity changes is beneficial, since it keeps the model simple while it simultaneously allows KeYmaera X to look for unusual corner cases.

The robot follows the same control as in Model 2 but includes differential equations for the obstacle. The main difference to Model 2 is the *safe* condition (23), which now has to account for the fact that obstacles may move according to (24) while the robot tries to avoid collision. The difference of Model 3 compared to Model 2 is highlighted in **boldface**.

Identification of Safe Controls The most critical element is again the formula $safe_{\text{ps}}$ that control choices need to satisfy in order to always keep the robot safe. We extend the intuitive explanation from static safety to account for the additional obstacle terms in (23), again considering the extreme case where the radius $r = \infty$ is infinitely large and the robot, thus, travels on a straight line. The robot must account for the additional impact over the static safety margin (12) from the motion of the obstacle. During the stopping time ($\varepsilon + \frac{s+A\varepsilon}{b}$) entailed by (11) and (12), the obstacle might approach the robot, e. g., on a straight line with maximum velocity V to the point of collision:

$$V\left(\varepsilon + \frac{s+A\varepsilon}{b}\right) = V\left(\frac{s}{b} + \left(\frac{A}{b}+1\right)\varepsilon\right)\ . \tag{25}$$

The safety distance chosen for $safe_{\text{ps}}$ in (23) of Model 3 is the sum of the distances (11), (12), and (25). The safety proof will have to show that this construction was safe and that it is also safe for all other curved trajectories that the obstacle and robot could be taking instead.

Verification The robot in Model 3 is safe, if it maintains positive distance $\|p-o\| > 0$ to the obstacle while the robot is driving (see Table 3):

$$\psi_{\text{ps}} \equiv s \neq 0 \rightarrow (\|p-o\| > 0)\ . \tag{26}$$

In order to guarantee ψ_{ps}, the robot must stay at a safe distance, which still allows the robot to brake to a complete stop before the approaching obstacle reaches the robot. The following condition captures this requirement as an

invariant φ_{ps} that we prove to hold for all loop executions:

$$\varphi_{\text{ps}} \equiv s \neq 0 \rightarrow \left(\|p - o\| > \frac{s^2}{2b} + V\frac{s}{b} \right) \quad . \tag{27}$$

Formula (27) says that, while the robot is driving, the positions of the robot and the obstacle are safely apart. This accounts for the robot's braking distance $\frac{s^2}{2b}$ while the obstacle is allowed to approach the robot with its maximum velocity V in time $\frac{s}{b}$. We prove that formula (26) holds for all executions of Model 3 when started in a non-collision state as for static safety, i. e., $\phi_{\text{ps}} \equiv \phi_{\text{ss}}$ (15).

Theorem 2. Passive safety. *Robots following Model 3 will never collide with static or moving obstacles while driving, as expressed by the provable* dL *formula* $\phi_{ps} \rightarrow [dw_{ps}]\psi_{ps}$.

Proof. The KeYmaera X proof uses invariant φ_{ps} (27). It extends the differential invariants (16)–(20) for static safety with invariants (28)–(29) about obstacle motion. Similar to the robot, the obstacle does not leave its bounding square of half side length tV around its previous position old(o).

$$-tV \leq o_x - \text{old}(o_x) \leq tV \tag{28}$$
$$-tV \leq o_y - \text{old}(o_y) \leq tV \tag{29}$$

□

Passive Friendly Safety of Obstacle Avoidance

In this section, we explore the stronger requirements of passive friendly safety, where the robot not only stops safely itself, but also allows for the obstacle to stop before a collision occurs. Passive friendly safety requires the robot to take careful decisions that respect the dynamic capabilities of moving obstacles. The intuition behind passive friendly safety is that our own robot should retain enough space for other obstacles to stop. Unlike passive safety, passive friendly safety ensures that there will not be collisions, as long as every obstacle makes a corresponding effort to avoid collision when it sees the robot, even when some obstacles approach intersections carelessly and turn around corners without looking. The definition of Maček et al. (2009) requires that the robot respects the worst-case braking time of the obstacle, which depends on its velocity and control capabilities. In our model, the worst-case braking time is a consequence of the following assumptions. We assume an upper bound τ on the obstacle's reaction time and a lower bound b_o on its braking capabilities. Then, τV is the maximal distance that the obstacle can travel before beginning to react and $\frac{V^2}{2b_o}$ is the maximal distance for the obstacle to stop from the maximal velocity V with an assumed minimum braking capability b_o.

Modeling Model 4 uses the same basic obstacle avoidance algorithm as Model 3. The difference is reflected in what the robot considers to be a safe distance to an obstacle. As shown in (31) the safe distance not only accounts for the robot's own braking distance, but also for the braking distance $\frac{V^2}{2b_o}$ and reaction time τ of the obstacle. The verification of passive friendly safety is more complicated than passive safety as it accounts for the behavior of the obstacle discussed below.

Model 4 Dynamic window with passive friendly safety

$$dw_{\text{pfs}} \equiv \left(ctrl_o; ctrl_r(a := A\ ,\ safe_{\text{pfs}}); dyn_{\text{ps}}\right)^* \tag{30}$$

$$safe_{\text{pfs}} \equiv \|p - o\|_\infty > \frac{s^2}{2b} + V\frac{s}{b} + \frac{\boldsymbol{V^2}}{\boldsymbol{2b_o}} + \boldsymbol{\tau V} + \left(\frac{A}{b} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(s + V)\right) \tag{31}$$

In Model 4 the obstacle controller $ctrl_o$ is a coarse model given by equation (22) from Model 3, which only constrains its non-negative velocity to be less than or equal to V. Such a liberal obstacle model is useful for analyzing the robot, since it requires the robot to be safe even in the presence of rather sudden obstacle behavior (e. g., be safe even if driving behind an obstacle that stops instantaneously or changes direction radically). However, now that obstacles must avoid collision once the robot is stopped, such instantaneous behavior becomes too powerful. An obstacle that can stop or change direction instantaneously can trivially avoid collision, which would not tell us much about real vehicles that have to brake before coming to a stop. Here, instead, we consider a more interesting refined obstacle behavior with braking modeled similar to the robot's braking behavior by the hybrid program *obstacle* given in Model 5.

Model 5 Refined obstacle with acceleration control

$$obstacle \equiv (ctrl_{\tilde{o}};\ dyn_{\tilde{o}})^* \tag{32}$$

$$ctrl_{\tilde{o}} \equiv a_o := *;\ ?v + a_o\tau \leq V \tag{33}$$

$$dyn_{\tilde{o}} \equiv t := 0;\ \{t' = 1,\ o' = vd_o, v' = a_o\ \&\ t \leq \tau \wedge v \geq 0\} \tag{34}$$

The refined obstacle may choose any acceleration a_o, as long as it does not exceed the velocity bound V (33). In order to ensure that the robot does not force the obstacle to avoid collision by steering (e. g., other cars at an intersection should not be forced to change lanes), we keep the obstacle's direction unit vector d_o

constant. The dynamics of the obstacle are straight ideal-world translational motion in the two-dimensional plane with reaction time τ, see (34).

Verification We verify the safety of the robot's control choices as modeled in Model 4. Unlike the passive safety case, the passive friendly safety property ϕ_{pfs} should guarantee that if the robot stops, moving obstacles (cf. Model 5) still have enough time and space to avoid a collision. The conditions $v = \sqrt{v_x^2 + v_y^2} \wedge d_{ox}v = v_x \wedge d_{oy}v = v_y$ link the combined velocity and direction vector (v_x, v_y) of the abstract obstacle model from the robot safety argument to the velocity scalar v and direction unit vector (d_{ox}, d_{oy}) of the refined obstacle model in the liveness argument. This requirement can be captured by the following dL formula:

$$\begin{aligned}\eta_{\text{pfs}} \equiv \big(\eta_{\text{obs}} \wedge 0 \leq v \wedge v = \sqrt{v_x^2 + v_y^2} \wedge \\ vd_{ox} = v_x \wedge vd_{oy} = v_y\big) \rightarrow \\ \langle \mathit{obstacle} \rangle \, (\|p - o\| > 0 \wedge v = 0)\end{aligned} \tag{35}$$

where the property η_{obs} accounts for the stopping distance of the obstacle:

$$\eta_{\text{obs}} \equiv \|p - o\| > \frac{V^2}{2b_o} + \tau V \ .$$

Formula (35) says that there exists an execution of the hybrid program *obstacle*, (existence of a run is formalized by the diamond operator $\langle \mathit{obstacle} \rangle$ in dL), that allows the obstacle to stop ($v = 0$) without having collided ($\|p - o\| > 0$). Passive friendly safety ψ_{pfs} is now stated as

$$\psi_{\text{pfs}} \equiv (s \neq 0 \rightarrow \eta_{\text{obs}}) \wedge \eta_{\text{pfs}} \ .$$

We study passive friendly safety with respect to initial states satisfying the following property:

$$\phi_{\text{pfs}} \equiv \eta_{obs} \wedge r \neq 0 \wedge \|d\| = 1 \ . \tag{36}$$

Observe that, in addition to the condition η_{pfs}, the difference to passive safety is reflected in the special treatment of the case $s = 0$. Even if the robot starts with speed $s = 0$ (which is passively safe), η_{obs} must be satisfied to prove passive friendly safety, since otherwise the obstacle may initially start out too close and thus unable to avoid collision. Likewise, we are required to prove η_{pfs} as part of ψ_{pfs} to guarantee that obstacles can avoid collision after the robot came to a full stop.

Theorem 3. Passive friendly safety. *Robots following Model 4 will never collide while driving and will retain sufficient safety distance for others to avoid collision, as expressed by the provable* dL *formula* $\phi_{pfs} \rightarrow [dw_{pfs}]\psi_{pfs}$.

Proof. The proof in KeYmaera X splits into a safety argument for the robot and a liveness argument for the obstacle. The loop and differential invariants in the robot safety proof are similar in spirit to passive safety, but account for the additional obstacle reaction time and stopping distance $\frac{V^2}{2b_o} + \tau V$. The obstacle liveness proof bases on *loop convergence*, i. e., it uses conditions that describe how much progress the loop body of the hybrid program *obstacle* can make towards stopping. Intuitively, the obstacle has made sufficient progress if either it is stopped already or can stop by braking n times:

$$v - n\tau b_o \leq 0 \vee v = 0 \ .$$

Additionally, the convergence conditions include the familiar bounds on the parameters ($\|d_o\| = 1$, $b_o > 0$, $\tau > 0$, and $0 \leq v \leq V$) and the remaining stopping distance $\|p - o\| > \frac{v^2}{2b_o}$. □

The symbolic bounds on velocity, acceleration, braking, and time in the above models represent uncertainty implicitly (e. g., the braking power b can be instantiated with the minimum specification of the robot's brakes, or with the actual braking power achievable w.r.t. the current terrain). Whenever knowledge about the current state is available, the bounds can be instantiated more aggressively to allow efficient robot behavior. For example, in a rare worst case we may face a particularly fast obstacle, but right now there are only slow-moving obstacles around. Or the worst case reaction time ε may be slow when difficult obstacle shapes are computed, but is presently quick as circular obstacles suffice to find a path. Theorems 1–3 are verified for all those values. Section Arbitrary Number of Obstacles illustrates how to explicitly model different kinds of obstacles simultaneously in a single model. Other aspects of uncertainty need explicit changes in the models and proofs, as discussed in subsequent sections.

Passive Orientation Safety

So far, we did not consider orientation as part of the safety specification. The notion of passive safety requires the robot to stop to avoid imminent collision, which can be inefficient or even impossible when sensor coverage is not exhaustive. For example, if an obstacle is close behind the robot (cf. Fig. 4), the robot would have to stop to obey passive safety. This may be the right behavior in an unstructured environment like walking pedestrians but is not helpful when driving on the lanes of a road. With a more liberal safety notion, the robot could choose a new curve that leads away from the obstacle.

We introduce *passive orientation safety* that only requires the robot to remain safe with respect to the obstacles in its *orientation of responsibility*. Overall system safety depends on the sensor coverage of the robot and the obstacles. For

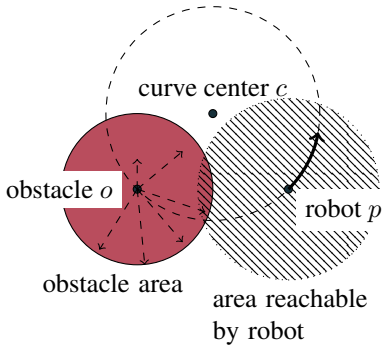


Figure 4. When ignoring orientation, passive safety requires the robot to stop when the robot's reachable area and the trajectory overlap with the obstacle area, even when moving away would increase the safety distance.

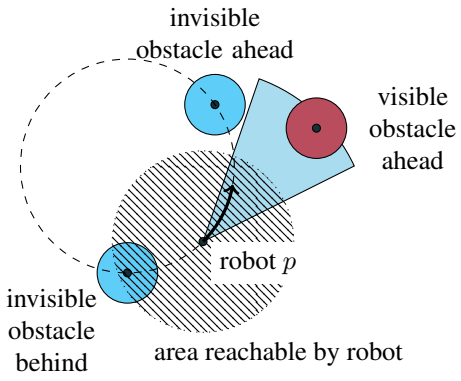


Figure 5. Passive orientation safety: The area observable by the robot (circular sector centered at robot): the distance to all visible obstacles must be safe. The robot must also ensure that it can stop inside its current observable area, since an obstacle might sit just outside the observable area. Obstacles outside the visible area are responsible for avoiding collision with the robot until they become visible, i. e., obstacles are assumed to not blindside the robot.

example, if two robots drive side-by-side with only very narrow sensor coverage to the front, they might collide when their paths cross. Even with limited sensor coverage, if both robots can observe some separation markers in space (e. g., lane markers) that keeps their paths separated, then passive orientation safety ensures that there will not be collisions. Likewise, passive orientation safety ensures that there will be no collisions when every robot and obstacle covers $180°$ in its orientation of responsibility, i. e., everyone is responsible for obstacles ahead, but not for those behind.

This notion of safety is suitable for structured spaces where obstacles can easily determine the trajectory and observable region of the robot (e. g., lanes on streets). The robot is responsible for collisions inside its observable area ("field of vision", cf. Fig. 5) and has to ensure that it can stop if needed before leaving the observable region, because it could otherwise cause collisions when moving into the blind spot just outside its observable area.

The robot does not make guarantees for obstacles that it cannot see. If an obstacle starts outside the observable region and subsequently hits the robot, then it is considered the fault of the obstacle. If the robot guarantees passive orientation safety and every obstacle outside the observable region guarantees that it will not interfere with the robot, a collision between the robot and an obstacle never happens while the robot is moving. In fact, collisions can be avoided when obstacles do not cross the trajectory of the robot. Any obstacles inside the observable region can drive with passive safety restrictions (i. e., guarantee not to exceed a maximum velocity) because the robot will brake or choose a new curve to avoid collisions. Obstacles that start outside the observable region can rely on the robot to only enter places it can see (i. e. the robot will be able to stop before it drives to places that it did not see when evaluating the safety of a curve).

Modeling To express that an obstacle was invisible to the robot when it chose a new curve, in Model 6 we introduce a variable *Visible* with *Visible* > 0 indicating that an obstacle was visible to the robot when it chose a new curve. The observable region is aligned with the orientation of the robot and extends symmetrically to the left and right of the orientation vector d by a constant design parameter γ that captures the angular width of the field of vision. The robot can see everything within angle $\frac{\gamma}{2}$ to its left or right. With these, passive orientation safety can be expressed as:

$$\psi_{\text{pos}} \equiv s \neq 0 \rightarrow \|p - o\| > 0 \vee (\mathit{Visible} \leq 0 \wedge |\beta| < \gamma)$$

This means that, when the robot is driving ($s \neq 0$), every obstacle is either sufficiently far away or it came from outside the observable region (so *Visible* ≤ 0) while the robot stayed inside $|\beta| < \gamma$. For determining whether or not the robot stayed inside the observable region, we compare the robot's angular progress β along the curve with the angular width γ of the observable region, see Fig. 6 for details. The angular progress β is reset to zero when the robot chooses a new curve in (37) and evolves according to $\beta' = \omega$ when the robot moves (40). Thus, β always holds the value of the angle on the current curve between the current position of the robot and its position when it chose the curve. Passive safety is a special case of passive orientation safety for $\gamma = \infty$. The model does not take advantage of the fact that $360°$ already subsumes unrestricted visibility. Passive orientation safety restricts admissible curves to those where the robot can stop before $|\beta| > \gamma$.

The new robot controller now only takes obstacles in its observable region into account (modeled by variable *Visible*

Model 6 Passive orientation safety

$$dw_{\text{pos}} \equiv \left(ctrl_o; ctrl_r(a := A; \boldsymbol{\beta := 0; Visible := *}\ ,\ safe_{\text{pos}} \wedge \boldsymbol{cda}); dyn_{\text{pos}}\right)^* \tag{37}$$

$$safe_{\text{pos}} \equiv \boldsymbol{Visible > 0 \rightarrow} \|p-o\|_\infty > \frac{s^2}{2b} + V\frac{s}{b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(s+V)\right) \tag{38}$$

$$\boldsymbol{cda \equiv \gamma|r| > \frac{s^2}{2b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon s\right)} \tag{39}$$

$$dyn_{\text{pos}} \equiv t := 0;\ \{p_r' = sd,\ d_r' = \omega d^\perp,\ v_r' = a,\ \boldsymbol{\beta' = \omega},\ \omega_r' = \frac{a}{r},\ p_o' = v,\ t' = 1\ \&\ s \geq 0 \wedge t \leq \varepsilon\} \tag{40}$$

to distinguish between obstacles that the sensors can see and those that are invisible) when computing the safety of a new curve in $safe_{\text{pos}}$ (38). In an implementation of the model, *Visible* is naturally represented since sensors only deliver distances to visible obstacles anyway. It chooses curves such that it can stop before leaving the observable region, i. e., it ensures a clear distance ahead (*cda*): such a curve is characterized by the braking distance of the robot being less than $\gamma|r|$, which is the length of the arc between the starting position when choosing the curve and the position where the robot would leave the observable region, cf. Fig. 6. In the robot's drive action (37) for selecting a new curve, the angular progress β along the curve is reset and the status of the obstacle (i. e. whether or not it is visible) is stored in variable *Visible* so that the visibility state is available when checking the safety property.

Verification Passive orientation safety (Theorem 4) is proved in KeYmaera X.

Theorem 4. Passive orientation safety. *Robots following Model 6 will never collide with the obstacles in sight while driving, and will never drive into unobservable areas, as expressed by the provable* dL *formula* $\phi_{pos} \rightarrow [dw_{pos}]\psi_{pos}$.

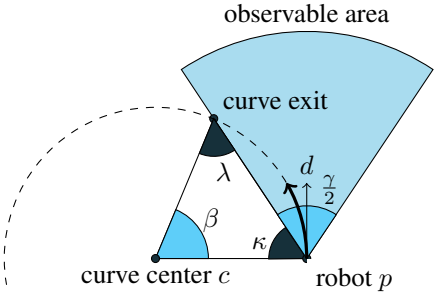


Figure 6. Determining the point where the curve exits the observable region of angular width γ by keeping track of the angular progress β along the curve: $\kappa = 90^\circ - \frac{\gamma}{2}$ because γ extends equally to both sides of the orientation d, which is perpendicular to the line from the robot to c (because d is tangential to the curve). $\lambda = \kappa$ because the triangle is isosceles. Thus, $\beta = 180^\circ - \kappa - \lambda = \gamma$ at exactly the moment when the robot would leave the observable region.

Proof. The proof in KeYmaera X extends the loop invariant conditions for passive safety so that the robot not only maintains the familiar stopping distance $\frac{s^2}{2b}$ to all obstacles, but also to the border of the visible region in case the nearest obstacle is invisible:

$$s > 0 \rightarrow \quad \|p-o\|_\infty > \frac{s^2}{2b} \\ \vee\ \mathit{Visible} \leq 0 \wedge |r\gamma| - |r\beta| > \frac{s^2}{2b}\ .$$

Here, we characterize the angular progress β with the differential invariant $\beta = \text{old}(\beta) + \frac{1}{r}\left(\text{old}(s)t + \frac{a}{2}t^2\right)$, in addition to the differential invariants for passive safety used in the proof of Theorem 2. □

Refined Models for Safety Verification

The models used for safety verification so far made simplifying assumptions to focus on the basics of different safety notions. In this section, we discuss how to create more realistic models with different accelerations, measurement uncertainty, actuator disturbance, asynchronous control of obstacle and robot, and explicit representation of arbitrary many obstacles. We introduce the model extensions for passive safety (Model 3) as an example. The extensions apply to static safety and passive friendly safety in a similar fashion by adapting $safe_{\text{ss}}$ and $safe_{\text{pfs}}$; passive orientation safety needs to account for the changes both in the translational safety margin $safe_{\text{pos}}$ and the angular progress *cda*.

Passive Safety with Actual Acceleration

Model 3 uses the robot's maximum acceleration A in its safety requirement (23) when it determines whether or not a new curve will be safe. This condition is conservative, since the robot of Model 3 can only decide between maximum acceleration ($a := A$) or maximum braking ($a := -b$ from Model 1). If (23) does not hold (which is independent from the chosen curve, i. e. the radius r), then Model 3 forces a

driving robot to brake with maximum deceleration $-b$, even if it might be sufficiently safe to coast or slightly brake or just not accelerate in full. As a result, Model 3 is passively safe but lacks efficiency in that it may take the robot longer to reach a goal because it can only decide between extreme choices. Besides efficiency concerns, extreme choices are undesirable for comfort reasons (e. g., decelerating a car with full braking power should be reserved for emergency cases).

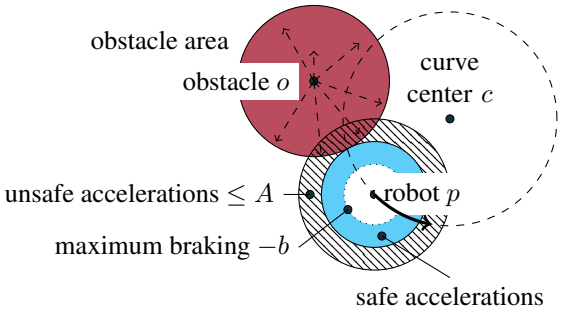


Figure 7. Passive safety with actual acceleration: the actual acceleration choice $-b \leq a \leq A$ must not take the robot into the area reachable by the obstacle. Dotted circle around robot position p: earliest possible stop with maximum braking $-b$; solid blue area between dotted circle and dashed area: safe a; dashed area: reachable with unsafe accelerations.

Fig. 7 illustrates how safety constraint (23) represents the maximally conservative choice: it forces the robot to brake (the outermost circle around the robot p intersects with the obstacle), even though many points reachable with $-b \leq a < A$ would have been perfectly safe (solid blue area does not intersect with the obstacle).

Modeling Model 7 refines Model 3 to work with the actual acceleration, i. e., in the acceleration choice (41) the robot picks any arbitrary acceleration a within the physical limits $-b \leq a \leq A$ instead of just maximum acceleration.

Model 7 Passive safety with actual acceleration

$$dw_{\text{psa}} \equiv \big(ctrl_o; ctrl_r(\boldsymbol{a := *; ? -b \leq a \leq A}, safe_{\text{psa}}); dyn_{\text{ps}}\big)^* \tag{41}$$

$$safe_{\text{psa}} \equiv \|p - o\|_\infty > \begin{cases} \boldsymbol{dist_{\geq}} & \textbf{if } \boldsymbol{s + a\varepsilon \geq 0} \\ \boldsymbol{dist_{<}} & \textbf{otherwise} \end{cases} \tag{42}$$

This change requires us to adapt the control condition (42) that keeps the robot safe. We first give the intuition behind condition (42), then justify its correctness with a safety proof.

Identification of Safe Constraints Following Loos et al. (2013b) we relax constraint (23) so that the robot can choose any acceleration $-b \leq a \leq A$ and checks this actual acceleration a for safety. That way, it only has to fall back to the emergency braking branch $a := -b$ if there is no other safe choice available. We distinguish two cases:

- $s + a\varepsilon \geq 0$: the acceleration choice $-b \leq a \leq A$ always keeps a nonnegative velocity during the full cycle duration ε.
- $s + a\varepsilon < 0$: the acceleration choice $-b \leq a < 0$ cannot be followed for the full duration ε without stopping the evolution to prevent a negative velocity.

In the first case, we continue to use formula (23) with actual a substituted for A to compute the safety distance:

$$dist_{\geq} = \frac{s^2}{2b} + V\frac{s}{b} + \left(\frac{a}{b} + 1\right)\left(\frac{a}{2}\varepsilon^2 + \varepsilon(s + V)\right) \tag{43}$$

In the second case, distance (43) is unsafe, because the terminal velocity when following a for ε time is negative (unlike in case 1). Thus, the robot may have collided at a time before ε, while the term in (43) only indicates that it will no longer be in a collision state at time ε after having moved backwards. Consider the time t_b when the robot's velocity becomes zero ($s + at_b = 0$) so that its motion stops (braking does not make the robot move backwards but merely stop). Hence, $t_b = -\frac{s}{a}$ since case 1 covers $a = 0$. Within duration t_b the robot will drive a total distance of $dist_r = -\frac{s^2}{2a} = \int_0^{t_b} s + at\,dt$. The obstacle may drive up to $dist_o = Vt_b$ until the robot is stopped. Thus, we compute the distance using (44) to account for the worst case that both robot and obstacle drive directly towards each other (note that $-b \leq a < 0$).

$$dist_{<} = -\frac{s^2}{2a} - V\frac{s}{a} \tag{44}$$

Verification We verify the safety of the actual acceleration control algorithm as modeled in Model 7 in KeYmaera X.

Theorem 5. Passive safety with actual acceleration. *Robots following Model 7 to base their safety margins on the current acceleration choice instead of worst-case acceleration will never collide while driving, as expressed by the provable dL formula* $\phi_{ps} \rightarrow [dw_{psa}]\psi_{ps}$.

Even though the safety constraint $safe_{\text{psa}}$ now considers the actual acceleration instead of the maximum possible acceleration when estimating the required safety margin, it can still be conservative when the robot makes sharp turns. During sharp turns, the straight-line distance from the origin is shorter than the distance along the circle, which can be exploited when computing the safety margin. This extension is in Appendix Passive Safety for Sharp Turns.

Passive Safety Despite Uncertainty

Robots have to deal with uncertainty in almost every aspect of their interaction with the environment, ranging from sensor inputs (e. g., inaccurate localization, distance measurement) to actuator effects (e. g., uncertain wheel slip depending on the terrain). In this section, we show how the three most important classes of uncertainty can be handled explicitly in the models. First, we allow localization uncertainty, so that the robot knows its position only approximately, which has a considerable impact on uncertainty over time. We then consider imperfect actuator commands, which means that the effective physical braking and acceleration will differ from the controller's desired output. Finally, we allow velocity uncertainty, so the robot knows its velocity only approximately, which also has an impact over time. We use nondeterministic models of uncertainty as intervals around the real position, acceleration, and velocity, without any probabilistic assumptions about their distribution.[g] Such intervals are instantiated, e. g., according to sensor or actuator specification (e. g., GPS error), or w.r.t. experimental measurements.[h]

Location Uncertainty Model 8 introduces location uncertainty. It adds a location measurement $\hat{p}$ before the control decisions are made such that the controller only bases its decisions on the most recent location measurement $\hat{p}$, which can deviate from the true location p. This location measurement may deviate from the real position p by no more than the symbolic parameter $\Delta_p \geq 0$, cf. (46). The measured location $\hat{p}$ is used in all control decisions of the robot (e. g., in (47) to compute whether or not it is safe to change the curve). The robot's physical motion still follows the real position p even if the controller does not know it.

Model 8 Passive safety despite location uncertainty, extends Model 3

$$dw_{\text{pslu}} \equiv \big(\boldsymbol{locate}; ctrl_o; ctrl_r(a := A\ ,\ safe_{\text{pslu}}); dyn_{\text{ps}}\big)^* \tag{45}$$

$$\boldsymbol{locate} \equiv \boldsymbol{\hat{p} := (*,*);\ ?\|\hat{p} - p\| \leq \Delta_p} \tag{46}$$

$$safe_{\text{pslu}} \equiv \|\boldsymbol{\hat{p}} - o\|_\infty > \frac{s^2}{2b} + V\frac{s}{b} + \boldsymbol{\Delta_p} + \left(\frac{A}{b} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(s+V)\right) \tag{47}$$

Theorem 6. Passive safety despite location uncertainty. *Robots computing their safety margins from location measurements with maximum uncertainty Δ_p by Model 8 will never collide while driving, as expressed by the provable* dL *formula* $\phi_{ps} \wedge \Delta_p \geq 0 \rightarrow [dw_{pslu}]\psi_{ps}$.

Uncertainty about the obstacle's position is already included in the nondeterministic behavior of previous models by increasing the shapes according to uncertainty.

Actuator Perturbation Model 9 introduces actuator perturbation between control and dynamics, cf. (48). Actuator perturbation affects the acceleration by a damping factor δ_a, known to be at most a maximum damping Δ_a, i. e., $\delta_a \in [\Delta_a, 1]$, cf. (49). Note that the damping factor δ_a can change arbitrarily often, but is assumed to be constant during the continuous evolution that takes a maximum of ε time units. The perturbation may cause the robot to now have full acceleration ($\delta_a = 1$) but later fully reduced braking ($\delta_a = \Delta_a$). This combination results in the largest possible stopping distance (for a certain speed s). For instance, the robot accelerates on perfect terrain, but is unlucky enough to be on slippery terrain again when it needs to brake. The robot considers this worst-case scenario during control in its safety constraint (50).

Model 9 Passive safety despite actuator perturbation, extends Model 3

$$dw_{\text{psap}} \equiv \big(ctrl_o; ctrl_r(a := A\ ,\ safe_{\text{psap}}); \boldsymbol{act}; dyn_{\text{ps}}\big)^* \tag{48}$$

$$\boldsymbol{act \equiv \delta_a := *;\ ?(0 < \Delta_a \leq \delta_a \leq 1);\ \tilde{a} := \delta_a a} \tag{49}$$

$$safe_{\text{psap}} \equiv \|p - o\|_\infty > \frac{s^2}{2b\boldsymbol{\Delta_a}} + V\frac{s}{b\boldsymbol{\Delta_a}} + \left(\frac{A}{b\boldsymbol{\Delta_a}} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(s+V)\right) \tag{50}$$

dyn_{ps} of Model 3 with a replaced by disturbed $\boldsymbol{\tilde{a}}$

Theorem 7. Passive safety despite actuator perturbation. *Robots with inaccurate actuation being subject to maximum disturbance Δ_a according to Model 9 will never collide while driving, as expressed by the provable* dL *formula*

$$\phi_{ps} \wedge \Delta_a > 0 \rightarrow [dw_{psap}]\psi_{ps} \ .$$

Velocity Uncertainty Model 10 introduces velocity uncertainty. To account for the uncertainty, at the beginning of its control phase the robot reads off a (possibly inexact) measurement $\hat{s}$ of its speed s. It knows that the measured speed $\hat{s}$ deviates by at most a measurement error Δ_s from

[g] Other error models are supported, as long as they are clipped to guaranteed intervals, because in the safety proof we have to analyze all measured values, regardless of their probability. For an advanced analysis technique considering probabilities, see stochastic dL Platzer (2011a).

[h] Instantiation with probabilistic bounds means that the symbolically guaranteed safety is traded for a probability of safety.

Model 10 Passive safety despite velocity uncertainty, extends Model 3

$$dw_{\text{psvu}} \equiv \left(\boldsymbol{sense}; ctrl_o; ctrl_r(a := A\ ,\ safe_{\text{psvu}}); dyn_{\text{ps}}\right)^* \tag{51}$$

$$\boldsymbol{sense} \equiv \hat{s} := *;\ ?(\hat{s} \geq 0 \wedge s - \boldsymbol{\Delta_s} \leq \hat{s} \leq s + \boldsymbol{\Delta_s}) \tag{52}$$

$$safe_{\text{psvu}} \equiv \|p - o\|_\infty > \frac{(\hat{s} + \boldsymbol{\Delta_s})^2}{2b} + V\frac{\hat{s} + \boldsymbol{\Delta_s}}{b} + \left(\frac{A}{b} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(\hat{s} + \boldsymbol{\Delta_s} + V)\right) \tag{53}$$

the actual speed s, see (52). Also, the robot knows that its speed is non-negative. Thus, we can assume that $\hat{s}$ is always equal to or greater than zero by transforming negative measurements. In order to stay safe, the controller has to make sure that the robot stays safe even if its true speed is maximally larger than the measurement, i. e. $s = \hat{s} + \Delta_s$. The idea is now that the controller makes all control choices with respect to the maximal speed $\hat{s} + \Delta_s$ instead of the actual speed s. The continuous evolution, in contrast, still uses the actual speed s, because the robot's physics will not be confused by a sensor measurement error.

Since we used the maximal possible speed when considering the safety of new curves in the controller we can prove that the robot will still be safe. A modeling subtlety arises when using $\hat{s}$ instead of s in the second branch (52) of $ctrl_r$: Because of the velocity uncertainty we no longer know if s is zero (i. e. the robot is stopped). However, the branch for stopped situations models discrete physics rather than a conscious robot decision (even if a real robot controller chooses to hit the brakes, as soon as the robot is stopped physics turns this decision into $a = 0$), so we still use the test $?(s = 0)$ instead of $?(\hat{s} = 0)$.

Theorem 8. Passive safety despite velocity uncertainty. *Robots computing their safety margins from velocity measurements with maximum uncertainty Δ_v according to Model 10 will never collide while driving, as expressed by the provable* dℒ *formula* $\phi_{ps} \wedge \Delta_v \geq 0 \rightarrow [dw_{psvu}]\psi_{ps}$.

Asynchronous Control of Obstacle and Robot

In the models so far, the controllers of the robot and the obstacle were executed synchronously, i. e., the robot and the obstacle made their control decisions at the same time. While the obstacle could always choose its previous control choices again if it does not want to act, the previous models only allowed the obstacle to decide when the robot made a control decision, too.[i] This does not reflect reality perfectly, since we want liberal obstacle models without assumptions about when an obstacle makes a control decision. So, we ensure that the robot remains safe regardless of how often and at which times the obstacles change their speed and orientation.

Model 11 Asynchronous obstacle and robot control, extends Model 3

$$dw_{\text{psns}} \equiv \left(ctrl_r(a := A\ ,\ safe_{\text{ps}});\ t := 0;\ \left(\boldsymbol{ctrl_o};\ \boldsymbol{dyn_{\text{ps}}}\right)^*\right)^* \tag{54}$$

In Model 11 we now model the control of the obstacle $ctrl_o$ in an inner loop around the continuous evolution dyn in (54) so that the obstacle control can interrupt continuous evolution at any time to make a decision, and then continue the dynamics immediately without giving the robot's controller a chance to run. This means that the obstacle can make as many control decisions as it wants without the robot being able to react every time. The controller $ctrl_r$ of the robot is still guaranteed to be invoked after at most time ε has passed, as modeled with the evolution domain constraint $t \leq \varepsilon$ in dyn_{ps}.

Theorem 9. Passive safety for asynchronous controllers. *Robots following Model 11 will never collide while driving, even if obstacles change their direction arbitrary often and fast, as expressed by the provable* dℒ *formula* $\phi_{ps} \rightarrow [dw_{psns}]\psi_{ps}$.

Proof. The KeYmaera X proof of Theorem 9 uses ϕ_{ps} as an invariant for the outer loop, whereas the invariant for the inner loop additionally preserves the differential invariants used for handling the dynamics dyn_{ps}. □

Arbitrary Number of Obstacles

The safety proofs so far modeled obstacles with a sensor system that nondeterministically delivers the position of any obstacle, including the nearest obstacle, to the control algorithm. In this section, we also explicitly analyze how that sensor system lets the robot avoid collision with each one of many obstacles. In order to prevent duplicating variables for each of the objects, which is undesirable even for a very small, known number of objects, we need a way of referring to countably many objects concisely.

Quantified Differential Dynamic Logic With *quantified differential dynamic logic* QdℒPlatzer (2010c, 2012d), we can explicitly refer to each obstacle individually by using quantification over objects of a sort (here all objects of the

[i] Note that dℒ follows the common assumption that discrete actions do not take time; time only passes in ODEs. So all discrete actions happen at the same real point in time, even though they are ordered sequentially.

sort O of obstacles). QdL is an extension of dL suited for verifying distributed hybrid systems by quantifying over sorts. QdL extends hybrid programs to *quantified hybrid programs*, which can describe the dynamics of distributed hybrid systems with any number of agents. Instead of using a single state variable $o_x : \mathbb{R}$ to describe the x coordinate of one obstacle, we can use a function term $o_x : O \to \mathbb{R}$ in QdL to denote that obstacle i has x-coordinate $o_x(i)$, for each obstacle i of obstacle sort O. Likewise, instead of a single two-dimensional state variable $o : \mathbb{R}^2$ to describe the planar position of one obstacle, we can use a function term $o : O \to \mathbb{R}^2$ in QdL to denote that obstacle i is at position $o(i)$, for each obstacle i. We use a *non-rigid* function symbol o, which means that the value of all $o(i)$ may change over time (e. g., the position $o(car)$ of an obstacle named *car*). Other function symbols are *rigid* if they do not change their values over time (e. g., the maximum velocity $V(i)$ of obstacle i never changes). Pure differential dynamic logic dL uses the sort $\mathbb{R}$. QdL formulas can use quantifiers to make statements about all obstacles of sort O with $\forall i \in O$ and $\exists i \in O$, similar to the quantifiers for the special sort $\mathbb{R}$ that dL already provides.

QdL allows us to explicitly track properties of all obstacles simultaneously. Of course, it is not just the position data that is important for obstacles, but also that the model allows all moving obstacles to change their positions according to their respective differential equations. Quantified hybrid programs allow the evolution of properties expressed as non-rigid functions for all objects of the same sort simultaneously (so all obstacles move simultaneously).

Table 4 lists the additional statements that quantified hybrid programs add to those of hybrid programs Platzer (2010c, 2012d).

Table 4. Statements of quantified hybrid programs

Statement	Effect
$\forall i{\in}C\, x(i) := \theta$	Assigns the current value of term θ to $x(i)$ simultaneously for all objects of sort C.
$\forall i{\in}C\, \big(x(i)' = \theta(i) \;\&\; Q\big)$	Evolves all $x(i)$ for any i along differential equations $x(i)' = \theta(i)$ restricted to evolution domain Q

We can use QdL to look up characteristics of specific obstacles, such as their maximum velocity, which allows an implementation to react to different kinds of obstacles differently if appropriate sensors are available.

Modeling In Model 12 we move from hybrid programs to quantified hybrid programs for distributed hybrid systems Platzer (2010c, 2012d), i. e., systems that combine distributed systems aspects (lots of obstacles) with hybrid systems aspects (discrete control decisions and continuous motion). We introduce a sort O representing obstacles so that arbitrarily many obstacles can be represented in the model simultaneously. Each obstacle i of the sort O has a maximum velocity $V(i)$, a current position $o(i)$, and a current vectorial velocity $v(i)$. We use non-rigid function symbols $o : O \to \mathbb{R}^2$, $v : O \to \mathbb{R}^2$, and $V : O \to \mathbb{R}$. Both $o(i)$ and $v(i)$ are two-dimensional vectors.

This new modeling paradigm also allows for another improvement in the model. So far, an arbitrary obstacle was chosen by picking any position nondeterministically in $ctrl_r$. Such a nondeterministic assignment includes the closest one. A controller implementation needs to compute which obstacle is actually the closest one (or consider them all one at a time). Instead of assigning the closest obstacle nondeterministically in the model, QdL can consider all obstacles by quantifying over all obstacles of sort O.

In the obstacle controller $ctrl_o$ (56) we use a loop to allow multiple obstacles to make a control decision. Each run of that loop selects one obstacle instance i arbitrarily and updates its velocity vector (but no longer its position, since obstacles are now tracked individually). The loop can be repeated arbitrarily often, so any arbitrary finite number of obstacles can make control choices in (56). In the continuous evolution, we quantify over *all obstacles* i of sort O in order to express that all obstacles change their state simultaneously according to their respective differential equations (58).

Initial condition, safety condition, and loop invariants are as before (26)–(27) except that they are now phrased for all obstacles $i \in O$. Initially, our robot is assumed to be stopped and we do not need to assume anything about the obstacles initially because passive safety does not consider collisions when stopped:

$$\phi_{\text{nobs}} \equiv s = 0 \land r \neq 0 \land \|d\| = 1 \tag{59}$$

The safety condition is passive safety for all obstacles:

$$\psi_{\text{nobs}} \equiv s \neq 0 \to \forall i{\in}O\, \|p - o(i)\|_\infty > 0 \tag{60}$$

Verification We use QdL to prove passive safety in the presence of arbitrarily many obstacles. Note that the controller condition $safe_{\text{nobs}}$ for multiple obstacles needs to distinguish obstacles that will stop during the next control cycle from those that will not.

Theorem 10. Passive safety for arbitrarily many obstacles. *Robots tracking any number of obstacles of their respective maximum velocities by Model 12 will never collide with any obstacle while driving, as expressed by the provable* QdL *formula* $\phi_{nobs} \to [dw_{nobs}]\psi_{nobs}$.

Proof. Since QdL is not yet implemented in KeYmaera X, we proved Theorem 10 with its predecessor KeYmaera. The

Model 12 Explicit representation of countably many obstacles, extends Model 7

$$dw_{\text{nobs}} \equiv \big(ctrl_o;\ ctrl_r(a := *; ?(-b \le a \le A)\ ,\ safe_{\text{nobs}});\ dyn_{\text{nobs}}\big)^* \tag{55}$$

$$ctrl_o \equiv \big(\boldsymbol{i := *};\ \boldsymbol{v(i)} := (*,*);\ \boldsymbol{?\|v(i)\| \le V(i)}\big)^* \tag{56}$$

$$safe_{\text{nobs}} \equiv \boldsymbol{\forall i \in O}\ \|p - \boldsymbol{o(i)}\|_\infty > \begin{cases} \boldsymbol{-\frac{s^2}{a} - V(i)\frac{s}{a}} & \textbf{if } \boldsymbol{s + a\varepsilon < 0} \\ \frac{s^2}{2b} + \boldsymbol{V(i)}\frac{s}{b} + \left(\frac{a}{b}+1\right)\left(\frac{a}{2}\varepsilon^2 + \varepsilon(s + \boldsymbol{V(i)})\right) & \text{otherwise} \end{cases} \tag{57}$$

$$dyn_{\text{nobs}} \equiv \boldsymbol{\forall i \in O}\ (t' = 1,\ p' = sd,\ d' = -\omega d^\perp,\ s' = a,\ \omega' = \frac{a}{r},\ \boldsymbol{o'(i) = v(i)}\ \&\ s \ge 0 \wedge t \le \varepsilon) \tag{58}$$

proof uses (27) with explicit $\forall i \in O$ as loop invariant:

$$\varphi_{\text{nobs}} \equiv s \neq 0 \to \forall i \in O\ \|p - o(i)\|_\infty > \frac{s^2}{2b} + V(i)\frac{s}{b}$$

The proof uses quantified differential invariants to prove the properties of the quantified differential equations Platzer (2011b). □

Liveness Verification of Ground Robot Navigation

Safety properties formalize that a precisely-defined bad behavior (such as collisions) will never happen. Liveness properties formalize that certain good things (such as reaching a goal) will ultimately happen. It is easy to design a trivial controller that is only safe (just never moves) or only live (full speed toward the goal ignoring all obstacles). The trick is to design robot controllers that meet both goals. The safe controllers identified in the previous sections guarantee safety (no collisions) and still allow motion. This combination of guaranteed safety under all circumstances (by a proof) and validated liveness under usual circumstances (validated only by some tests) is often sufficient for practical purposes. Yet, without a liveness proof, there is no guarantee that the robot controller will reach its respective goal except in the circumstances that have been tested before. In this section, we verify liveness properties, since the precision gained by formalizing the desired liveness properties as well as the circumstances under which they can be guaranteed are insightful.

Formalizing liveness properties is even more difficult and the resulting questions in practice much harder than safety (even if liveness can be easier in theory Platzer (2015)). Both safety and liveness properties only hold when they are true in the myriad of situations with different environmental behavior that they conjecture. They are diametrically opposed, because liveness requires motion but safety considerations inhibit motion. For the safe robot models that we consider here, liveness is, thus, quite a challenge, because there are many ways that environmental conditions or obstacle behavior would force the robot to stop or turn around for safety reasons, preventing it from reaching its goal. For example, an unrestricted obstacle could move around to block the robot's path and then, as the robot re-plans to find another trajectory, dash to block the new path too. To guarantee liveness, one has to characterize *all necessary conditions* that allow the robot to reach its goal, which are often prohibitively many. Full adversarial behavior can be handled but is challenging Platzer (2015).

For a liveness proof, we deem three conditions important:

Adversarial behavior. Carefully defines acceptable adversarial behavior that the robot can handle. For example, sporadically crossing a robot's path might be acceptable in the operating conditions, but permanently trapping the robot in a corner might not.

Conflicting goals. Identifies conflicting goals for different agents. For example, if the goal of one robot is to indefinitely occupy a certain space and that of another is to reach this very space it is impossible for both to satisfy their respective requirements.

Progress. Characterizes progress formally. For example, in the presence of obstacles, a robot sometimes needs to move away from the goal in order to ultimately get to the goal. But how far is a robot allowed to deviate on the detour?

Liveness properties that are actually true need to define some reasonable restrictions on the behavior of other agents in the environment. For example, a movable obstacle may block the robot's path for some limited amount of time, but not indefinitely. And when the obstacle moves on, it may not turn around immediately again. Liveness conditions might define a compromise between reaching the goal and having at least invested *reasonable effort* of trying to get to the goal, if unacceptable adversarial behavior occurred or goals conflicted, or progress is physically impossible.

In this section, we start with a stationary environment, so that we first can concentrate on finding a notion for progress for the robot itself. Next, we let obstacles cross the robot's path and define what degree of adversarial behavior is acceptable for guaranteeing liveness.

Reach a Waypoint on a Straight Lane

As a first liveness property, we consider a stationary environment without obstacles, which prevents adversarial behavior as well as conflicting goals, so that we can concentrate on the conditions to describe how the robot makes progress without the environment interfering. We focus on low-level motion planning where the robot has to make decisions about acceleration and braking in order to drive to a waypoint on a straight line. We want our robot to *provably* reach the waypoint, so that a high-level planning algorithm knows that the robot will reliably execute its plan by stitching together the complete path from straight-line segments between waypoints. To model the behavior at the final waypoint when the robot stops (because it reached its goal) and at intermediate waypoints in a uniform way, we consider a simplified version where the robot has to stop at each waypoint, before it turns toward the next waypoint. That way, we can split a path into straight-line segments that make it easier to define progress, because they are describable with solvable differential equations when abstracted into one-dimensional space.

Modeling We say that the robot reached the waypoint when it stops inside a region of size $2\Delta_g$ around the waypoint. That is: *(i)* at least one execution enters the goal region, and *(ii)* all executions stop before exiting the goal region $g+\Delta_g$. The liveness property ψ_{wp} (61) characterizes these conditions formally.

$$\psi_{\text{wp}} \equiv \langle dw_{\text{wp}}\rangle(g-\Delta_g < p) \wedge [dw_{\text{wp}}](p < g+\Delta_g) \quad (61)$$

Remark 1. *The liveness property ψ_{wp} (61) is formulated as a conjunction of two formulas: at least one run enters the goal region $\langle dw_{wp}\rangle\, g-\Delta_g < p$, while none exit the goal region on the other end $[dw_{wp}]\, p < g+\Delta_g$. In particular, there is a run that will stop inside the goal region, which, explicitly, corresponds to extending formula (61) to the following liveness property:*

$$\langle dw_{wp}\rangle\big(g-\Delta_g < p \wedge 0 \le s \le V_g \wedge \langle dw_{wp}\rangle\, s = 0\big) \wedge [dw_{wp}](p < g+\Delta_g) \quad (68)$$

Formula (68) means that there is an execution of model dw_{wp} where the robot enters the goal region without exceeding the maximum approach velocity V_g, and from where the model has an execution that will stop the robot $\langle dw_{wp}\rangle\, s = 0$. The proof for formula (68) uses the formula $s = 0 \vee (s > 0 \wedge s - n\varepsilon b \le 0)$ to characterize progress (i. e., braking for duration $n\varepsilon$ will stop the robot).

Model 13 describes the behavior of the robot for approaching a goal region. In addition to the three familiar options from previous models of braking unconditionally (63), staying stopped (64), or accelerating when safe (65), the model now contains a fourth control option (66) to slowly approach the goal region, because nondeterministically big acceleration choices might overshoot the goal.

The liveness proof has to show that the robot will get to the goal under *all* circumstances except those explicitly characterized as being assumed not to happen, e. g., unreasonably small goal regions, high robot velocity, or hardware faults, such as engine or brake failure. Similar to safety proofs, these assumptions are often linked. For example, what makes a goal region unreasonably small depends on the robot's braking and acceleration capabilities. The robot cannot stop at the goal if accelerating just once from its initial position will already make it impossible for the robot to brake before shooting past the goal region. In this case, both options of the robot will violate our liveness condition: it can either stay stopped and not reach the goal, or it can start driving and miss the goal.

Therefore, we introduce a maximum velocity V_g that the robot has to obey when it is close to the goal. That velocity must be small enough so that the robot can stop inside the goal region and is used as follows. While obeying the approach velocity V_g outside the goal region (66), the robot can choose any acceleration that will not let it exceed the maximum approach velocity. The dynamics of the robot in this model follows a straight line, assuming it is already oriented directly towards the goal (67).

Identification of Live Controls Now that we know what the goal of the robot is, we provide the intuition behind the conditions that make achieving the goal possible. The robot is only allowed to adapt its velocity with controls other than full braking when those controls will not overshoot the goal region, see $g+\Delta_g$ in (65) and $g-\Delta_g$ in (66). Condition $-b \le a \le \frac{V_g - s}{\varepsilon} \le A$ in (66) ensures that the robot will only pick acceleration values that will never exceed the approach velocity V_g in the next ε time units, i. e., until it can revise its decision. Once inside the goal region, the only remaining choice is to brake, which makes the robot stop reliably in the waypoint region.

The robot is stopped initially ($s = 0$) outside the goal region ($p < g-\Delta_g$), its brakes $b > 0$ and engine $A > 0$ are working,[j] and it has some known reaction time $\varepsilon > 0$:

$$\phi_{\text{wp}} \equiv s = 0 \wedge p < g-\Delta_g \wedge b > 0 \wedge A > 0 \wedge \varepsilon > 0 \wedge 0 < V_g \wedge V_g\varepsilon + \frac{V_g^2}{2b} < 2\Delta_g \quad (69)$$

Most importantly, the approach velocity V_g and the size of the goal region $2\Delta_g$ must be compatible. That way, we know that the robot has a chance to approach the goal with a velocity that fits to the size of the goal region.

[j] For safety, $A \ge 0$ was sufficient, but in order to reach a goal the robot must be able to accelerate to non-zero velocities.

Model 13 Robot follows a straight line to reach a waypoint

$$dw_{\text{wp}} \equiv (ctrl; dyn)^* \tag{62}$$
$$ctrl \equiv \quad (a := -b) \tag{63}$$
$$\cup\, (?s = 0;\ a := 0) \tag{64}$$
$$\cup\, (?p + \frac{s^2}{2b} + \left(\frac{A}{b} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon s\right) < g + \Delta_g \wedge s + A\varepsilon \leq V_g;\ a := A) \tag{65}$$
$$\cup\, (?p \leq g - \Delta_g \wedge s \leq V_g;\ a := *;\ ? - b \leq a \leq \frac{V_g - s}{\varepsilon} \leq A) \tag{66}$$
$$dyn \equiv t := 0;\ \{p' = s,\ s' = a,\ t' = 1\ \&\ t \leq \varepsilon \wedge s \geq 0\} \tag{67}$$

Verification Similar to safety verification, for liveness verification we combine the initial condition ϕ_{wp} (69), the model dw_{wp} (Model 13), and the liveness property ψ_{wp} (61) in Theorem 11.

Theorem 11. Reach waypoint. *Robots following Model 13 can reach the goal area $g - \Delta_g < p$ and will never overshoot $p < g + \Delta_g$, as expressed by the provable* dL *formula $\phi_{wp} \rightarrow \psi_{wp}$, i. e., with ψ_{wp} from (61) expanded:*

$$\phi_{wp} \rightarrow \big(\langle dw_{wp}\rangle(g - \Delta_g < p) \wedge [dw_{wp}](p < g + \Delta_g)\big)\ .$$

Proof. We proved Theorem 11 using KeYmaera X. Instead of an invariant characterizing what does not change, we now need a variant characterizing what it means to make progress towards reaching the goal region Platzer (2008, 2012a). If the progress measure indicates the goal would be reachable with n iterations of the main loop of Model 13, then we have to show that by executing the loop once we can get to a state where the progress measure indicates the goal would be reachable in the remaining $n - 1$ loop iterations.

Informally, the robot reaches the goal if it has a positive speed $s > 0$ and can enter the goal region by just driving for time $n\varepsilon$ with that speed, as summarized by the loop variant $\varphi_{\text{wp}} \equiv 0 < s \leq V_g \wedge g - \Delta_g < p + n\varepsilon s$. □

After having proved how the robot can always reach its goal when it is on its own, we next analyze liveness in the presence of other moving agents.

Cross an Intersection

In this section, we prove liveness for scenarios in which the robot has to pass an intersection, while a moving obstacle may cross the robot's path, so that the robot may need to stop for safety reasons to let the obstacle pass. We want to prove that it is always possible for the robot to successfully pass the intersection. The model captures the general case of a point-intersection with two entering roads and two exits at the opposing side, so that it subsumes any scenario where a robot and an obstacle drive straight to cross an intersection, as illustrated in Fig. 8.

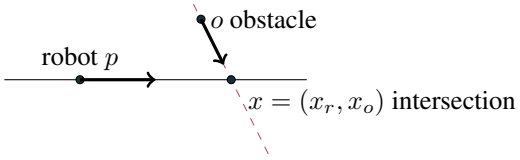


Figure 8. Illustration of the paths of a robot (black solid line) and an obstacle (red dashed line) crossing an intersection at point x.

Modeling Since there is a moving obstacle, the robot needs to follow a collision avoidance protocol in order to safely cross the intersection. We choose passive safety for simplicity. Collision avoidance alone, however, will not reliably let the robot make progress. Thus, we will model a robot that favors making progress towards the other side of the intersection, and only falls back to collision avoidance when the obstacle is too close to pass safely.

Intersections enable the obstacle to trivially prevent the robot from ever passing the intersection. All that the obstacle needs to do is just block the entire intersection forever by stopping there (e. g., somebody built a wall so that the intersection disappeared). Clearly, no one could demand the robot passes the intersection in such impossible cases. We prove that the robot can pass the intersection when obstacles *behave reasonably*, for a precisely defined characterization of what is reasonable for an obstacle to do. We, therefore, include a restriction on how long the obstacle may reside at the intersection. We choose a strictly positive minimum velocity V_{min} to prevent the obstacle from stopping. Other fairness conditions (e. g., an upper bound on how long the intersection can be blocked, enforced with a traffic light) are representable in hybrid programs as well.

Identification of Live Controls For ensuring progress, the model uses three conditions (*AfterX*, *PassFaster*, and

Model 14 Robot safely crosses an intersection

$$dw_{\text{cx}} \equiv (ctrl_o;\ ctrl_r;\ dyn)^* \tag{70}$$

$$ctrl_o \equiv a_o := *;\ ?(-b \le a_o \le A) \tag{71}$$

$$ctrl_r \equiv \begin{cases} a := *;\ ?(-b \le a \le A) & \text{if } \mathit{AfterX} \\ a := *;\ ?(0 \le a \le A) & \text{if } \mathit{PassFaster} \\ a := 0 & \text{if } \mathit{PassCoast} \\ \quad (a := -b) & \\ \cup\ (?s = 0;\ a := 0) & \text{otherwise Model 3} \\ \cup\ (?\mathit{safe};\ \ldots) & \end{cases} \tag{72}$$

$$dyn \equiv t := 0;\ \{p_r' = s,\ v_r' = a, \tag{73}$$

$$p_o' = v,\ v_o' = a_o,\ t' = 1 \tag{74}$$

$$\&\ t \le \varepsilon \wedge s \ge 0 \wedge v \ge V_{min}\} \tag{75}$$

PassCoast) that tell the robot admissible conditions for choosing its acceleration, depending on its own position and the obstacle position in relation to the intersection. The robot can choose any acceleration after it passed the intersection ($p > x_r$) or after the obstacle passed ($o > x_o$):

$$\mathit{AfterX} \equiv p > x_r \vee o > x_o \ .$$

The robot is allowed to increase its speed if it manages to pass safely in front of the obstacle (even if the obstacle speeds up during the entire process), or if speeding up would still let the robot pass safely behind the obstacle (even if the obstacle drives with only minimum speed V_{min}):

$$\mathit{PassFaster} \equiv s > 0 \wedge (\mathit{PassFront} \vee \mathit{PassBehind})$$

$$\mathit{PassFront} \equiv o + v\frac{x_r - p}{s} + \frac{A}{2}\left(\frac{x_r - p}{s}\right)^2 < x_o$$

$$\mathit{PassBehind} \equiv x_o < o + V_{min}\frac{x_r - p}{s + A\varepsilon}$$

The robot is allowed to just maintain its speed if it either passes safely in front or behind the obstacle with that speed:

$$\mathit{PassCoast} \equiv s > 0 \wedge x_o < o + V_{min}\frac{x_r - p}{s} \ .$$

In all other cases, the robot has to follow the collision avoidance protocol from Model 3 to choose its speed, modified accordingly for the one-dimensional variant here.

Verification As a liveness condition, we prove that the robot will make it past the intersection without colliding with the obstacle.

Theorem 12. Pass Intersection. *Robots following Model 14 can pass an intersection while avoiding collisions with obstacles at the intersection, as expressed in the provable* d$\mathcal{L}$ *formula*

$$\phi_{cx} \rightarrow \quad [dw_{cx}]\ (p = x_r \rightarrow o \neq x_o) \\ \wedge \langle dw_{cx} \rangle\ (p > x_r) \ .$$

Proof. We proved Theorem 12 in KeYmaera X. In the loop invariant of the safety proof we combine the familiar stopping distance $p + \frac{s^2}{2b} < x_r$ with the conditions *AfterX*, *PassCoast*, and *PassFront* that allow driving in its loop invariant:

$$0 \le s \wedge V_{\text{min}} \le v \wedge \big(p + \frac{s^2}{2b} < x_r \\ \vee \mathit{AfterX} \vee \mathit{PassCoast} \vee \mathit{PassFront}\big) \ .$$

The main insight in the liveness proof is that achieving the goal can be split into two phases: first, the robot waits for the obstacle to pass; afterwards, the robot accelerates to pass the intersection. We split the loop into these two phases with $\langle dw_{\text{cx}}{}^* \rangle (p > x_r) \leftrightarrow \langle dw_{\text{cx}}{}^* \rangle \langle dw_{\text{cx}}{}^* \rangle (p > x_r)$ so that we can analyze each of the resulting two loops with its own separate loop variant. In the first loop, we know that the obstacle drives with at least speed $v \ge V_{\text{min}}$, so with n steps it will pass the intersection, which is characterized in the loop variant $o + n\varepsilon V_{\text{min}} > x_o$. This loop variant implies $o > x_o$ when $n \le 0$. Once the obstacle is past the intersection, in the second loop the robot controller can safely favor its *AfterX* control. Since the robot might be stopped, we unroll the loop once to $\langle dw_{\text{cx}} \rangle \langle dw_{\text{cx}}{}^* \rangle (p > x_r)$ in order to ensure that the robot accelerates with A to a positive speed. The loop variant then exploits that the robot's speed is $s \ge A\varepsilon$ after accelerating once for time ε, so it will pass the intersection x_r with n steps of duration ε as follows: $p + n\varepsilon(A\varepsilon) > x_r$. □

The liveness proofs show that the robot *can* achieve a useful goal if it makes the right choices. When the robot controller is modeled such that it *always* makes the right choices, we prove that the controller will always safely make it to the goal within a specified time budget. We discuss robot controllers that provably meet deadlines in Appendix Liveness with Deadlines.

Monitoring for Compliance At Runtime

The previous sections discussed models of obstacle avoidance control and of the physical behavior of ground robots in their environment, and we proved that these models are guaranteed to possess crucial safety and liveness properties. The proofs present absolute mathematical evidence of the correctness of the models. If the models used for verification are an adequate representation of the real robot and its environment, these proofs transfer to the

real system. But any model necessarily deviates from the real system at least to some extent.

In this section, we discuss how to use ModelPlex Mitsch and Platzer (2016) to bridge the gap between models and reality by verification. The idea is to provably detect and safely respond to deviations between the model and the real robot in its environment by monitoring appropriate conditions at runtime. ModelPlex complements offline proofs with runtime monitoring. It periodically executes a *monitor*, which is systematically synthesized from the verified models by an automatic proof of correctness, and checks input from sensors and output to actuators for compliance with the verified model. If a deviation is detected, ModelPlex initiates a fail-safe action, e.g. stopping the robot or cutting its power to avoid actively running into obstacles, and, by that, ensure that *safety proofs* from the model carry over to the real robot. Of course, such fail-safe actions need to be triggered early enough to make sure the robot stops on time, which is what the monitors synthesized by ModelPlex ensure.

A monitor checks the actual evolution of the real robot implementation to discover failures and mismatches with the verified model. The acceleration chosen by the robot's control software implementation must fit to the current situation. For example, accelerate only when the verified model considers it safe. And the chosen curve must fit to the current orientation. No unintended change to the robot's speed, position, orientation has happened, and no violations of the assumptions about the obstacles have occurred. This means, any variable that is allowed to change in the model must be monitored. In the examples here, these variables include the robot's position p, longitudinal speed s, rotational speed ω, acceleration a, orientation d, curve r, obstacle position o and velocity v.

A ModelPlex monitor is designed for periodic sampling. For each variable there will be two observed values, one from the previous sample time (for example, previous robot position p) and one from the current sample time (for example, next robot position p^+). It is not important for ModelPlex that the values are measured exactly at the sampling period, but merely that there is an upper bound ε on the amount of time that passed between two samples. A ModelPlex monitor checks in a provably correct way whether the evolution observed in the difference of the sampled values can be explained by the model. If it does, the current behavior fits to a verified behavior and is, thus, safe. If it does not, the situation may have become unsafe and a fail-safe action is initiated to mitigate safety hazards.

Fig. 9 illustrates the principle behind a ModelPlex monitor. The values from the previous sample time serve as starting state for executing the model. The values produced by executing the model are then compared to the values observed in the current sample time by the monitor.

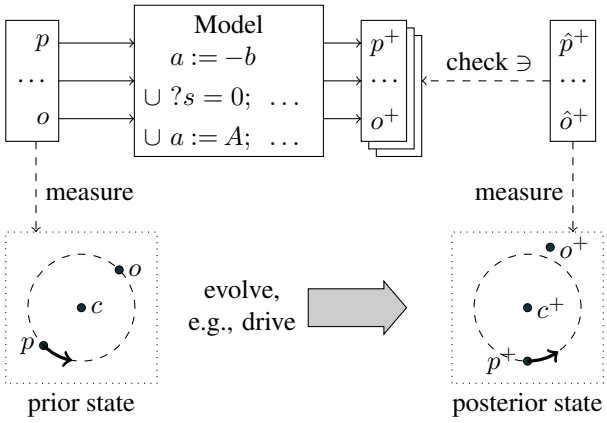


Figure 9. The principle behind a ModelPlex monitor: can the model reproduce or explain the observed real-world behavior?

The verified models themselves are too slow to execute, because they involve nondeterminism and differential equations. Hence, provably correct monitor expressions in real arithmetic are synthesized from a model using an offline proof in KeYmaera X. These expressions capture the behavior of the models, projected onto the pairwise comparisons of sampled values that are needed at runtime.

Monitor techniques for model compliance monitors and state prediction monitors are reported in Mitsch and Platzer (2016). Here, we focus on a controller monitor expression synthesized from Model 3, which captures all possible control decisions of the robot that are verified to be safe. The controller monitor checks the decisions of an (unverified) controller implementation for consistency with the verified discrete model without differential equations. ModelPlex automatically extracts the discrete model by a proof with the ordinary differential equation (ODE) being conservatively over-approximated by its evolution domain. The resulting condition *monitor* (76) in Fig. 10, which is synthesized by an automatic proof in KeYmaera X, mimics the structure of the model: it captures the assumptions on the obstacle mon_o, the evolution domain from dynamics mon_{dyn}, as well as the specification for each of the three controller branches (braking mon_b, staying stopped mon_s, or accelerating mon_a).

The obstacle monitor part mon_o in (77), checks that the measured obstacle velocity v^+ must not exceed the assumptions made in the model about their maximum velocity. The dynamics monitor part mon_{dyn} in (78) checks the evolution domain of the ODE and that the controller did reset its clock ($t^+ = 0$). The braking monitor mon_b in (79) defines that emergency brakes can only hit the brakes and do not change anything else (acceleration $a^+ = -b$, while everything else is of the form $x^+ = x$ meaning that no change is allowed). Monitor mon_s in (80) expresses that staying stopped is possible if the speed is zero ($s =$

$$monitor \equiv mon_o \wedge mon_{dyn} \wedge (mon_b \vee mon_s \vee mon_a) \tag{76}$$

$$mon_o \equiv \|v^+\| \leq V \tag{77}$$

$$mon_{dyn} \equiv 0 \leq \varepsilon \wedge s \geq 0 \wedge t^+ = 0 \tag{78}$$

$$mon_b \equiv o^+ = o \wedge p^+ = p \wedge d^+ = d \wedge s^+ = s \wedge \omega^+ = \omega \wedge a^+ = -b \wedge r^+ = r \tag{79}$$

$$mon_s \equiv s = 0 \wedge o^+ = o \wedge p^+ = p \wedge d^+ = d \wedge s^+ = s \wedge \omega^+ = 0 \wedge a^+ = 0 \wedge r^+ = r \tag{80}$$

$$mon_a \equiv a^+ = A \wedge r^+ \neq 0 \wedge \omega^+ r^+ = s \wedge p^+ = p \wedge d^+ = d \wedge s^+ = s \tag{81}$$

$$\wedge \|p - o^+\|_\infty > \frac{s^2}{2b} + V\frac{s}{b} + \left(\frac{A}{b} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(s + V)\right) \tag{82}$$

Figure 10. Synthesized safety conditions. The generated monitor captures conditions on obstacles mon_o, on dynamics mon_{dyn}, and on the robot's decisions on braking mon_b, staying stopped mon_s, and accelerating mon_a. The monitor distinguishes two observed values per variable, separated by a controller run (for example, p denotes the position before running the controller, whereas p^+ denotes the position after running the controller).

0) and the controller must have chosen no acceleration and no rotation ($a = 0$ and $\omega = 0$), while everything else is unchanged. Finally, the acceleration monitor mon_a in (81)–(82) says that, if the distance is safe, the robot can choose maximum acceleration $a^+ = A$, a new non-spinning steering $r^+ \neq 0$ that fits to the current speed $\omega^+ r^+ = s$; position, orientation, and speed must not be set by the controller (those follow from the acceleration and steering choice).

Conclusion and Future Work

Robots are modeled by hybrid systems, because they share continuous physical motion with advanced computer algorithms controlling their behavior. We demonstrate that this understanding also helps proving robots safe. We develop hybrid system models for collision avoidance algorithms for autonomous ground vehicles and prove that the algorithms guarantee static safety for static obstacles and both passive safety and passive friendly safety in the presence of moving obstacles.

We augment the models and safety proofs with robustness for localization uncertainty and imperfect actuation. Incremental revision of models and proofs helps reducing the verification complexity, since in lower-fidelity models the safety-critical effects of computations on the physical behavior are easier to predict and characterize in control conditions. Additional details can then be understood incrementally as extensions to these previously found control conditions (e. g., it helps to first understand the safety margins for perfect sensing, and later add the impact of uncertainty on the behavior of the robot). All parameters in our models—such as those for maximum obstacle velocity and sensor/actuator uncertainty—are fully symbolic and can be instantiated arbitrarily, including bounds from probabilistic models (e. g., assume the 2σ confidence interval of the distribution of obstacle velocities as maximum obstacle velocity). In this case, our verified safety guarantees translate into a probability of safety.

Theorems 1–9, 11, and 12 were proved with significant automation and succinct reusable proof tactics in the dℒ theorem prover KeYmaera X. All theorems were additionally proved in its predecessor KeYmaera. The most important insight in all proofs were the loop and differential invariants. Some proofs needed simple insights on how to eliminate variables to reduce the complexity of the resulting arithmetic.

Overall, the tactics follow a similar structure across all theorems, with only minor variation. The tactics use proof automation for symbolic execution of programs, for proving differential invariants, and for simplifying arithmetic, which all perform a large number of internal steps automatically to turn the proof hints provided by users into actual proofs. Table 5 summarizes the proof statistics: the tactic size characterizes the manual effort of the user (mostly proof hints on differential invariants and minor arithmetic simplifications), while the proof steps are the corresponding internal steps taken by KeYmaera X to fill in gaps in the proof hints with automated proof search and justify the proof from the axioms of dℒ. As a performance indicator, we list the total time needed to run the proofs on a 2.4GHz Intel Core i7 with 16GB memory, most of which is spent in external tools for handling real arithmetic with quantifier elimination (QE time column).

As part of the verification activity, we identified crucial safety constraints that have to be satisfied in order to choose a new curve or accelerate safely. These constraints are entirely symbolic and summarized in Table 6. The static safety invariant is equivalent to the admissible velocities identified in Fox et al. (1997), which assumes instantaneous control. Our proofs identified the invariants required

Table 5. Proof statistics in KeYmaera X

Theorem	Tactic size		Proof steps	Time $[s]$	
	LOC	Steps		QE	Total
Safety proofs					
1: Static	12	71	30355	74	89
2: Passive	12	73	51956	229	268
3: Passive-friendly	45	140	68620	342	407
4: Orientation	15	108	173989	934	1006
Passive safety extensions					
5: Acceleration	16	84	67604	405	463
6: Uncertain location	12	73	57775	445	485
7: Perturbation	21	120	56297	254	299
8: Uncertain velocity	12	94	54601	359	404
9: Async control	42	122	61772	284	335
Liveness proofs					
11: Reach waypoint	32	93	46530	69	125
12: Pass intersection	234	440	61878	83	182

for safety in the presence of moving obstacles, sensor uncertainty and coverage, and actuator perturbation, as well as the additional margins in column "safe control" that account for the reaction time of the robot. When instantiated with concrete numerical values of a robot design, these safety constraints can be used for design decision tradeoffs and to get an intuition about how conservative or aggressive our robot can drive, such as:

- how fast can the robot pass through a narrow door?
- how fast can the robot drive on a given corridor?

Appendix Interpretation of Verification Results analyzes the constraints for common values of acceleration force, braking force, control cycle time, and obstacle distance (i. e., door width, corridor width). These examples illustrate that the verified collision avoidance protocol is suitable for indoor navigation at reasonable speeds.

Future work includes exploiting more kinematic capabilities (e. g., going sideways with omni-drive) and explicit accounts for distance measurement uncertainty, which is, however, easier than location uncertainty.

Funding

This material is based upon work supported by NSF CAREER Award CNS-1054246, NSF EXPEDITION CNS-0926181, NSF CNS-1446712, by DARPA FA8750-12-2-0291, AFOSR FA9550-16-1-0288, and by Bosch. This project is funded in part by Carnegie Mellon University's Technologies for Safe and Efficient Transportation, the National USDOT University Transportation Center for Safety (T-SET UTC) which is sponsored by the US Department of Transportation. This work was also supported by the Austrian BMVIT under grant FIT-IT 829598, FFG BRIDGE 838526, and FFG Basisprogramm 838181.

Table 6. Invariant and safety constraint summary. Safe control margins account for the reaction time of the robot.

Safety		Invariant + Safe Control
Static	(Model 2, Theorem 1)	$\|p-o\|_\infty > \frac{s^2}{2b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon s\right)$
Passive	(Model 3, Theorem 2)	$s \neq 0 \rightarrow \|p-o\|_\infty > \frac{s^2}{2b} + V\frac{s}{b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(s+V)\right)$
Passive friendly	(Model 4+5, Theorem 3)	$\|p-o\|_\infty > \frac{s^2}{2b} + V\left(\frac{s}{b}+\tau\right) + \frac{V^2}{2b_o} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(s+V)\right)$
Passive orientation	(Model 6, Theorem 4)	*isVisible* $> 0 \rightarrow \|p-o\|_\infty > \frac{s^2}{2b} + V\frac{s}{b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(s+V)\right)$ and $\gamma\|r\| > \frac{s^2}{2b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon s\right)$
Extensions (passive safety examples)		
with actual acceleration	(Model 7, Theorem 5)	$s \neq 0 \rightarrow \|p-o\|_\infty > \begin{cases} -\frac{s^2}{2a} - V\frac{s}{a} & \text{if } s + a\varepsilon < 0 \\ \frac{s^2}{2b} + V\frac{s}{b} + \left(\frac{a}{b}+1\right)\left(\frac{a}{2}\varepsilon^2 + \varepsilon(s+V)\right) & \text{otherwise} \end{cases}$
+ location uncertainty	(Model 8, Theorem 6)	$s \neq 0 \rightarrow \|\hat{p}-o\|_\infty > \frac{s^2}{2b} + V\frac{s}{b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(s+V)\right) + \Delta_p$
+ actuator perturbation	(Model 9, Theorem 7)	$s \neq 0 \rightarrow \|p-o\|_\infty > \frac{s^2}{2b\Delta_a} + V\frac{s}{b\Delta_a} + \left(\frac{A}{b\Delta_a}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(s+V)\right)$
+ velocity uncertainty	(Model 10, Theorem 8)	$s \neq 0 \rightarrow \|p-o\|_\infty > \frac{(\hat{s}+\Delta_v)^2}{2b} + V\frac{\hat{s}+\Delta_v}{b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(\hat{s}+\Delta_v+V)\right)$
asynchronous robot and obstacle control	(Model 11, Theorem 9)	see Passive safety
arbitrary many obstacles	(Model 12, Theorem 10)	$s \neq 0 \rightarrow \forall i \in O\ \|p-o(i)\|_\infty > \frac{s^2}{2b} + V\frac{s}{b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(s+V(i))\right)$

References

Althoff D, Kuffner JJ, Wollherr D and Buss M (2012) Safety assessment of robot trajectories for navigation in uncertain and dynamic environments. *Auton. Robots* 32(3): 285–302. doi:10.1007/s10514-011-9257-9.

Bohrer B, Rahli V, Vukotic I, Völp M and Platzer A (2017) Formally verified differential dynamic logic. In: Bertot Y and Vafeiadis V (eds.) *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*. ACM, pp. 208–221. doi:10.1145/3018610.3018616.

Bonin-Font F, Ortiz A and Oliver G (2008) Visual navigation for mobile robots: A survey. *Journal of Intelligent and Robotic Systems* 53(3): 263–296. doi:10.1007/s10846-008-9235-4.

Bouraine S, Fraichard T and Salhi H (2012) Provably safe navigation for mobile robots with limited field-of-views in dynamic environments. *Auton. Robots* 32(3): 267–283. doi:10.1007/s10514-011-9258-8.

Bräunl T (2006) Driving robots. In: *Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems*. Springer, pp. 97–111.

Brock O and Khatib O (1999) High-speed navigation using the global dynamic window approach. In: *1999 IEEE International Conference on Robotics and Automation, Marriott Hotel, Renaissance Center, Detroit, Michigan, May 10-15, 1999, Proceedings*. pp. 341–346. doi:10.1109/ROBOT.1999.770002.

Choset H, Lynch K, Hutchinson S, Kantor G, Burgard W, Kavraki L and Thrun S (2005) *Principles Of Robot Motion*. MIT Press.

Collins GE (1975) Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: *Automata Theory and Formal Languages, 2nd GI Conference, Kaiserslautern, May 20-23, 1975*. pp. 134–183. doi:10.1007/3-540-07407-4_17.

Davenport JH and Heintz J (1988) Real quantifier elimination is doubly exponential. *J. Symb. Comput.* 5(1/2): 29–35. doi:10.1016/S0747-7171(88)80004-X.

Fiorini P and Prassler E (2000) Cleaning and household robots: A technology survey. *Auton. Robots* 9(3): 227–235. doi:10.1023/A:1008954632763.

Fiorini P and Shiller Z (1998) Motion planning in dynamic environments using velocity obstacles. *I. J. Robotics Res.* 17(7): 760–772. doi:10.1177/027836499801700706.

Fox D, Burgard W and Thrun S (1997) The dynamic window approach to collision avoidance. *IEEE Robot. Automat. Mag.* 4(1): 23–33. doi:10.1109/100.580977.

Frehse G, Guernic CL, Donzé A, Cotton S, Ray R, Lebeltel O, Ripado R, Girard A, Dang T and Maler O (2011) SpaceEx: Scalable verification of hybrid systems. In: Gopalakrishnan G and Qadeer S (eds.) *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, *LNCS*, volume 6806. Springer, pp. 379–395. doi:10.1007/978-3-642-22110-1_30.

Fulton N, Mitsch S, Bohrer B and Platzer A (2017) Bellerophon: Tactical Theorem Proving for Hybrid Systems. In: Ayala-Rincón M and Muñoz CA (eds.) *ITP*, *LNCS*, volume 10499. Springer. doi:10.1007/978-3-319-66107-0_14.

Fulton N, Mitsch S, Quesel JD, Völp M and Platzer A (2015) KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In: Felty AP and Middeldorp A (eds.) *CADE*, *LNCS*, volume 9195. Springer, pp. 527–538. doi:10.1007/978-3-319-21401-6_36.

Hart PE, Nilsson NJ and Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics* 4(2): 100–107. doi:10.1109/TSSC.1968.300136.

Karaman S and Frazzoli E (2012) Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications. In: *American Control Conference, ACC 2012, Montreal, QC, Canada, June 27-29, 2012*. IEEE, pp. 735–742.

Khatib O (1985) Real-time obstacle avoidance for manipulators and mobile robots. In: *Proceedings of the 1985 IEEE International Conference on Robotics and Automation, St. Louis, Missouri, USA, March 25-28, 1985*. pp. 500–505. doi:10.1109/ROBOT.1985.1087247.

Kress-Gazit H, Fainekos GE and Pappas GJ (2009) Temporal-Logic-Based Reactive Mission and Motion Planning. *IEEE Trans. Robotics* 25(6): 1370–1381. doi:10.1109/TRO.2009.2030225.

LaValle SM and Kuffner JJ (2001) Randomized kinodynamic planning. *I. J. Robotic Res.* 20(5): 378–400. doi:10.1177/02783640122067453.

Loos SM, Platzer A and Nistor L (2011) Adaptive cruise control: Hybrid, distributed, and now formally verified. In: Butler M and Schulte W (eds.) *FM*, *LNCS*, volume 6664. Springer, pp. 42–56. doi:10.1007/978-3-642-21437-0_6.

Loos SM, Renshaw DW and Platzer A (2013a) Formal verification of distributed aircraft controllers. In: Belta C and Ivancic F (eds.) *Proceedings of the 16th international conference on Hybrid systems: computation and control, HSCC 2013, April 8-11, 2013, Philadelphia, PA, USA*. ACM, pp. 125–130. doi:10.1145/2461328.2461350.

Loos SM, Witmer D, Steenkiste P and Platzer A (2013b) Efficiency analysis of formally verified adaptive cruise controllers. In: Hegyi A and Schutter BD (eds.) *ITSC*. pp. 1565–1570. doi:10.1109/ITSC.2013.6728453.

Maček K, Vasquez Govea DA, Fraichard T and Siegwart R (2009) Towards Safe Vehicle Navigation in Dynamic Urban Scenarios. *Automatika* 50(3–4): 184–194.

Minguez J, Montano L and Santos-Victor J (2006) Abstracting vehicle shape and kinematic constraints from obstacle avoidance methods. *Auton. Robots* 20(1): 43–59. doi:10.1007/s10514-006-5363-5.

Mitsch S, Ghorbal K and Platzer A (2013) On provably safe obstacle avoidance for autonomous robotic ground vehicles. In: Newman P, Fox D and Hsu D (eds.) *Robotics: Science and Systems*. ISBN 978-981-07-3937-9. doi:10.15607/RSS.2013.IX.014.

Mitsch S, Loos SM and Platzer A (2012) Towards formal verification of freeway traffic control. In: Lu C (ed.) *ICCPS*. IEEE. ISBN 978-0-7695-4695-7, pp. 171–180. doi:10.1109/ICCPS.2012.25.

Mitsch S and Platzer A (2016) ModelPlex: Verified runtime validation of verified cyber-physical system models. *Formal Methods in System Design* 49(1): 33–74. doi:10.1007/s10703-016-0241-z. Special issue of selected papers from RV'14.

Müller A, Mitsch S, Retschitzegger W, Schwinger W and Platzer A (2016) A component-based approach to hybrid systems safety verification. In: Ábrahám E and Huisman M (eds.) *Integrated Formal Methods - 12th International Conference, IFM 2016, Reykjavik, Iceland, June 1-5, 2016, Proceedings*, *LNCS*, volume 9681. Springer, pp. 441–456. doi:10.1007/978-3-319-33693-0_28.

Pan J, Zhang L and Manocha D (2012) Collision-free and smooth trajectory computation in cluttered environments. *I. J. Robotics Res.* 31(10): 1155–1175. doi:10.1177/0278364912453186.

Plaku E, Kavraki LE and Vardi MY (2009) Hybrid systems: from verification to falsification by combining motion planning and discrete search. *Formal Methods in System Design* 34(2): 157–182. doi:10.1007/s10703-008-0058-5.

Plaku E, Kavraki LE and Vardi MY (2013) Falsification of LTL safety properties in hybrid systems. *STTT* 15(4): 305–320. doi:10.1007/s10009-012-0233-2.

Platzer A (2008) Differential dynamic logic for hybrid systems. *J. Autom. Reas.* 41(2): 143–189. doi:10.1007/s10817-008-9103-8.

Platzer A (2010a) *Logical analysis of hybrid systems: Proving theorems for complex dynamics*. Springer. ISBN 978-3-642-14508-7. doi:10.1007/978-3-642-14509-4.

Platzer A (2010b) Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. Comput.* 20(1): 309–352. doi:10.1093/logcom/exn070.

Platzer A (2010c) Quantified differential dynamic logic for distributed hybrid systems. In: Dawar A and Veith H (eds.) *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, *LNCS*, volume 6247. Springer, pp. 469–483. doi:10.1007/978-3-642-15205-4_36.

Platzer A (2011a) Stochastic differential dynamic logic for stochastic hybrid programs. In: Bjørner N and Sofronie-Stokkermans V (eds.) *CADE*, *LNCS*, volume 6803. Springer, pp. 431–445. doi:10.1007/978-3-642-22438-6_34.

Platzer A (2011b) Quantified differential invariants. In: Frazzoli E and Grosu R (eds.) *HSCC*. ACM, pp. 63–72. doi:10.1145/1967701.1967713.

Platzer A (2012a) Logics of dynamical systems. In: *LICS*. IEEE. ISBN 978-1-4673-2263-8, pp. 13–24. doi:10.1109/LICS.2012.13.

Platzer A (2012b) The complete proof theory of hybrid systems. In: *LICS*. IEEE. ISBN 978-1-4673-2263-8, pp. 541–550. doi:10.1109/LICS.2012.64.

Platzer A (2012c) The structure of differential invariants and differential cut elimination. *Logical Methods in Computer Science* 8(4): 1–38. doi:10.2168/LMCS-8(4:16)2012.

Platzer A (2012d) A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems. *Logical Methods in Computer Science* 8(4): 1–44. doi:10.2168/LMCS-8(4:17)2012. Special issue for selected papers from CSL'10.

Platzer A (2015) Differential game logic. *ACM Trans. Comput. Log.* 17(1): 1:1–1:51. doi:10.1145/2817824.

Platzer A (2017) A complete uniform substitution calculus for differential dynamic logic. *J. Autom. Reas.* 59(2): 219–265. doi:10.1007/s10817-016-9385-1.

Platzer A and Clarke EM (2009) Formal verification of curved flight collision avoidance maneuvers: A case study. In: Cavalcanti A and Dams D (eds.) *FM*, *LNCS*, volume 5850. Springer, pp. 547–562. doi:10.1007/978-3-642-05089-3_35.

Platzer A and Quesel JD (2008) KeYmaera: A hybrid theorem prover for hybrid systems. In: Armando A, Baumgartner P and Dowek G (eds.) *IJCAR*, *LNCS*, volume 5195. Springer, pp. 171–178. doi:10.1007/978-3-540-71070-7_15.

Quesel JD, Mitsch S, Loos S, Aréchiga N and Platzer A (2016) How to Model and Prove Hybrid Systems with KeYmaera: A Tutorial on Safety. *STTT* 18(1): 67–91. doi:10.1007/s10009-015-0367-0.

Sarid S, Xu B and Kress-Gazit H (2012) Guaranteeing high-level behaviors while exploring partially known maps. In: *Robotics: Science and Systems VIII, University of Sydney, Sydney, NSW, Australia, July 9-13, 2012*. doi:10.15607/RSS.2012.VIII.048.

Seward DW, Pace C and Agate R (2007) Safe and effective navigation of autonomous robots in hazardous environments. *Auton. Robots* 22(3): 223–242. doi:10.1007/s10514-006-9721-0.

Sogokon A, Ghorbal K, Jackson PB and Platzer A (2016) A method for invariant generation for polynomial continuous systems. In: Jobstmann B and Leino KRM (eds.) *VMCAI*, *LNCS*, volume 9583. Springer, pp. 268–288. doi:10.1007/978-3-662-49122-5_13.

Täubig H, Frese U, Hertzberg C, Lüth C, Mohr S, Vorobev E and Walter D (2012) Guaranteeing functional safety: design for provability and computer-aided verification. *Auton. Robots* 32(3): 303–331. doi:10.1007/s10514-011-9271-y.

van den Berg J, Abbeel P and Goldberg KY (2011) LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. *I. J. Robotics Res.* 30(7): 895–913. doi:10.1177/0278364911406562.

Wolff EM, Topcu U and Murray RM (2014) Optimization-based trajectory generation with linear temporal logic specifications. In: *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*. IEEE, pp. 5319–5325. doi:10.1109/ICRA.2014.6907641.

Wu A and How JP (2012) Guaranteed infinite horizon avoidance of unpredictable, dynamically constrained obstacles. *Auton. Robots* 32(3): 227–242. doi:10.1007/s10514-011-9266-8.

Supplemental material

Passive Safety for Sharp Turns

Models 3 and 7 used a safety distance in supremum norm $\|\cdot\|_\infty$ for the safety constraints, which conservatively overapproximates the actual trajectory of the robot by a box around the robot. For example, recall the safety distance (23) of Model 3

$$\|p-o\|_\infty > \frac{s^2}{2b} + V\frac{s}{b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(s+V)\right) \tag{23*}$$

which needs to be large enough in either one axis, irrespective of the actual trajectory that the robot will be taking. This constraint is safe but inefficient when the robot chooses a trajectory that will keep it close to its current position (e. g., when driving along a small circle, meaning it makes a sharp turn). For example, a robot with constant velocity $s = 4$ and reaction time $\varepsilon = 1$ will traverse a small circle with radius $r = \frac{1}{\pi}$ and corresponding circumference $2\pi r = 2$ twice within time ε. Safety constraint (23) required the total distance of 4 as a safety distance between the robot and the obstacle, because it overapproximated its actual trajectory by a box. However, the robot never moves away more than $\frac{2}{\pi}$ from its original position then because it moves on a circle (cf. Fig. 11a). With full 360° sensor coverage the robot can exploit that the closest obstacle does not cross its trajectory, which makes this extension suitable for passive safety and passive friendly safety, but not for passive orientation safety.

Modeling We change the robot controller to improve its efficiency. One choice would be to explicitly express circular motion in terms of sine and cosine and then compute all possible positions of the robot explicitly. However, besides being vastly inefficient in a real controller, this introduces transcendental functions and would leave decidable real arithmetic. Hence, we will use the distance of the obstacle to the trajectory itself in the control conditions. Such a distance computation requires that we adapt the constraint *curve* to express the curve center explicitly in (89). So far, the curve was uniquely determined by the radius r and the orientation d of the robot. Now that we need the curve center explicitly for distance calculation to the obstacle, the controller chooses the curve center c such that:

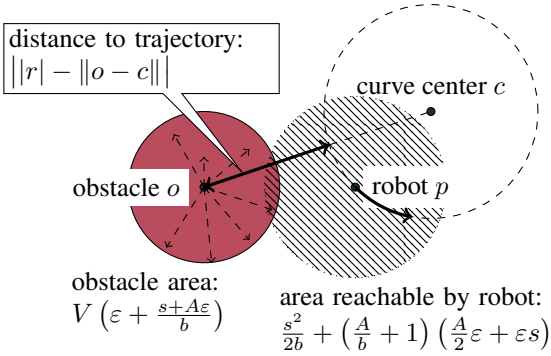
- $(p-c)$ is perpendicular to the robot orientation d, i. e., d is tangential to the curve, and
- $(p-c)$ is located correctly to the left or right of the robot, so that it fits to the clockwise or counter-clockwise motion indicated by the sign of r.

Thus, the condition *curve* (89) in Model 15 now checks if the chosen curve and the direction of the robot are consistent, i. e., $|r| = \|p-c\|$ and $d = \frac{(p-c)^\perp}{r}$. Additionally, we augment the robot with a capability to turn on the spot when stopped ($s = 0$).

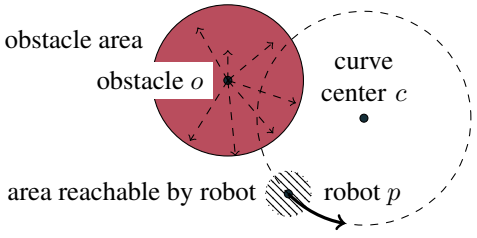
Model 15 Passive safety when considering the trajectory of the robot in distance measurement, extends Model 7

$$dw_{\text{psdm}} \equiv (ctrl_o; ctrl_r; dyn)^* \tag{83}$$
$$ctrl_o \equiv \text{see Model 3} \tag{84}$$
$$ctrl_r \equiv \quad (a := -b) \tag{85}$$
$$\cup \left(?s = 0;\ a := 0;\ w_r := 0;\ \ (d := -d \cup d := d)\,;\ r := *;\ c := (*,*);\ ?curve\right) \tag{86}$$
$$\cup (a := *;\ ? - b \leq a \leq A;\ \omega := *;\ ? - \Omega \leq \omega \leq \Omega; \tag{87}$$
$$r := *;\ c := (*,*);\ o := (*,*);\ ?curve \wedge safe) \tag{88}$$
$$curve \equiv r \neq 0 \wedge |r| = \|p - c\| \wedge d = \frac{(p-c)^\perp}{r} \wedge r\omega = s \tag{89}$$
$$safe \equiv \quad \left(\|p - o\|_\infty > \begin{cases} dist_{\geq} & \text{if } s + a\varepsilon \geq 0 \\ dist_{<} & \text{otherwise} \end{cases}\right) \tag{90}$$
$$\vee \left(\left|\,|r| - \|o - c\|\,\right| > \begin{cases} V\left(\varepsilon + \frac{s+a\varepsilon}{b}\right) & \text{if } s + a\varepsilon \geq 0 \\ -V\frac{s}{a} & \text{otherwise} \end{cases}\right)$$
$$dyn_{\text{psdm}} \equiv \text{see Model 3} \tag{91}$$

For this, (86) is extended with a choice of either turning around ($d := -d$) or remaining oriented as is ($d := d$) when stopped, and the corresponding choice of a curve center c such that the curve variables remain consistent according to the subsequent test $?curve$.



(a) Safe since the obstacle area does not overlap the dashed trajectory.



(b) Safe since obstacle area and dotted robot area do not overlap.

Figure 11. Two different reasons for safe robot trajectories

Identification of Safe Controls With the changes in distance measurement introduced above, we relax the control conditions that keep the robot safe. The distance of the obstacle to the trajectory can be described in two steps:

1. Calculate the distance of the obstacle to the circle: $\left|\,|r| - \|o - c\|\,\right|$, which is the absolute value of the radius minus the distance between the obstacle and the circle center.
2. Calculate the maximum distance that the obstacle can drive until the robot comes to a stop. This distance is equal to the distances calculated in the previous models, i. e. in the case $s + a\varepsilon \leq 0$ it is $-V\frac{s}{a}$ and in the case $s + a\varepsilon \geq 0$ it is $V\left(\varepsilon + \frac{s+a\varepsilon}{b}\right)$.

If the distance between the obstacle and the circle describing the robot's trajectory is greater than the sum of those distances, then the robot can stop before hitting the obstacle. Then choosing the new curve is safe, which leads us to choose the following safety condition:

$$\left|\,|r| - \|o - c\|\,\right| > \begin{cases} V\left(\varepsilon + \frac{s+a\varepsilon}{b}\right) & \text{if } s + a\varepsilon \geq 0 \\ -V\frac{s}{a} & \text{otherwise} \end{cases} \tag{92}$$

We use condition (92), which now uses the Euclidean norm $\|\cdot\|$, for choosing a new curve in Model 15. With this new constraint, the robot is allowed to choose the curve in Fig. 11a. However, constraint (92) has drawbacks when the trajectory of the robot is slow along a large circle and the obstacle is close to the circle, as illustrated in Fig. 11b. In this case the robot is only

allowed to choose very small accelerations because the obstacle is very close to the circle. Formula (90) in Model 15 follows the more liberal of the two constraints—i. e., (42) ∨ (92)—to provide the best of both worlds.

Verification We verify the safety of the robot's control choices in KeYmaera X.

Theorem 13. Passive safety with trajectory distance measurement. *Robots using trajectory distance measurement according to Model 15 in addition to direct distance measurement guarantee passive safety, as expressed by the provable* dℒ *formula* $\phi_{ps} \rightarrow [dw_{psdm}]\psi_{ps}$.

Proof. The most important condition in the loop invariant of the proof guarantees that the robot either maintains the familiar safe stopping distance $\|p-o\|_\infty > \frac{s^2}{2b}$, or that the obstacle cannot reach the robot's curve until the robot is stopped:

$$s > 0 \rightarrow \|p-o\|_\infty > \frac{s^2}{2b} \vee \big||r| - \|o-c\|\big| > V\frac{s}{b} \quad .$$

□

Liveness with Deadlines

The liveness proofs in the article showed that the robot can achieve a useful goal if it makes the right choices. The proofs neither guarantee that the robot will always make the right decisions, nor specify how long it will take until the goal will be achieved. In this section, we prove that it always achieves its goals within a given reasonable amount of time. Previously we showed that the robot *can* do the right thing to ultimately get to the goal, while here we prove that it *always* makes the right decisions that will take it to the waypoint or let it cross an intersection *within a bounded amount of time*. It is no longer enough to show existence of an execution that makes the robot achieve its goals. Now we need to show that all possible executions do so in the given time. This needs more deterministic controllers that only brake when necessary.

We are going to illustrate two alternatives for modeling arrival deadlines: in Section Reaching a Waypoint we use a countdown T that is initialized to the deadline and expires when $T \leq 0$, whereas in Section Crossing an Intersection we use T as a clock that is initialized to a starting value $T \leq 0$ and counts up to a deadline $D > 0$, so that two deadlines (crossing zero and exceeding D) can be represented with a single clock variable.

Reaching a Waypoint We start by defining a correctness condition for reaching a waypoint.

$$\psi_{\text{wpdl}} \equiv p < g + \Delta_g \wedge (T \leq 0 \rightarrow s = 0 \wedge g - \Delta_g < p) \quad (93)$$

Formula (93) expresses that the robot will never be past the goal region ($p < g + \Delta_g$), and after the deadline ($T \leq 0$, i. e. after countdown T expired) it will be stopped inside the goal region ($s = 0 \wedge g - \Delta_g < p$).

Modeling Model 16 is the familiar loop of control followed by dynamics (94). Unlike in previous models, braking and staying put is no longer allowed unconditionally for the sake of reaching the waypoint reliably in time (95). The robot accelerates maximally whenever possible without rushing past the waypoint region, cf. (95). In all other cases, the robot chooses acceleration to control towards the approach velocity V_g (95). The dynamics remain unchanged, except for the additional countdown $T' = -1$ of the deadline in (96).

Identification of Live Controls In order to prove this model live, we need to set achievable deadlines. The deadline has to be large enough *(i)* for the robot to accelerate to velocity V_g, *(ii)* drive to the waypoint with that velocity, and *(iii)* once it is there, have sufficient time to stop. It also needs a slack time ε, so that the robot has time to react to the deadline. Finally, the conditions ϕ_{wp} from (69), which enable the robot to reach a waypoint at all, have to hold as well. Formula (97) summarizes these deadline conditions.

$$\phi_{\text{wpdl}} \equiv \phi_{\text{wp}} \wedge T > \underbrace{\frac{V_g - s}{A}}_{(i)} + \underbrace{\frac{g - \Delta - p}{V_g}}_{(ii)} + \underbrace{\frac{V_g}{b}}_{(iii)} + \varepsilon \quad (97)$$

Verification A proof of the robot always making the right choices is a combination of a safety and a liveness proof: we have to prove that *all* choices of the robot reach the goal before the deadline expires (safety proof), and that *there exists at least one* way of the robot reaching the goal before the deadline expires (liveness proof). Both $[\cdot]$ and $\langle\cdot\rangle$ are needed to express that the robot always makes the right choices to get to the waypoint, since $[\cdot]$ alone does not guarantee existence of such a choice.

Theorem 14. Reach waypoint with deadline. *Robots following Model 16 will always reach the waypoint before the deadline expires, as expressed by the provable* dℒ *formula*

$$\phi_{wpdl} \rightarrow \big([dw_{wpdl}]\psi_{wpdl} \wedge \langle dw_{wpdl}\rangle\psi_{wpdl}\big) \quad .$$

Proof. We proved Theorem 14 with KeYmaera X, using automated tactics to handle the solvable differential equation system. The proof uses the following conditions as loop invariants:

$$p + \frac{s^2}{2b} < g + \Delta_g \wedge 0 \leq s \leq V_g \wedge$$
$$\begin{cases} s = 0 \vee T \geq \frac{s}{b} & \text{if } g - \Delta_g < p \\ T > \frac{g-\Delta_g-p}{A\varepsilon} + \frac{V_g}{b} + \varepsilon & \text{if } p \leq g - \Delta_g \wedge s \geq A\varepsilon \\ T > \varepsilon - \frac{s}{A} + \frac{g-\Delta_g-p}{A\varepsilon} + \frac{V_g}{b} + \varepsilon & \text{if } p \leq g - \Delta_g \wedge s \leq A\varepsilon \end{cases}$$

The robot maintains sufficient margin to avoid overshooting the goal area and it respects the approach velocity V_g. Reaching the goal is then split into increasingly critical cases: if the robot already is at the goal ($g - \Delta_g < p$) it is either stopped already or will manage to stop before the deadline expires. If the robot is not yet at the goal, but at least already traveling with some non-zero speed $s \geq A\varepsilon$, then it still has sufficient time to drive to the goal with the current speed and stop. Finally, if the robot is not yet traveling fast enough, it still has sufficient time to speed up. □

Model 16 Robot reaches a waypoint before a deadline

$$dw_{\text{wpdl}} \equiv (ctrl; dyn)^* \tag{94}$$

$$ctrl \equiv \begin{cases} (a := -b) \cup (?s = 0; a := 0) & \textbf{if } \boldsymbol{g - \Delta_g < p} \\ a := A & \textbf{if } \boldsymbol{p + \frac{s^2 - V_g^2}{2b} + \left(\frac{A}{b} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon s\right) \leq g - \Delta_g} \\ a := *; \ ? -b \leq a \leq \frac{V_g - s}{\varepsilon} \leq A & \text{otherwise} \end{cases} \tag{95}$$

$$dyn \equiv t := 0; \ p' = s, \ s' = a, \ t' = 1, \ \boldsymbol{T' = -1} \ \& \ t \leq \varepsilon \wedge s \geq 0 \tag{96}$$

Crossing an Intersection Crossing an intersection before a deadline is more complicated than reaching a waypoint, because the robot may need to wait for the intersection to clear so that the robot can cross it safely in the first place.

Modeling Model 17 remains almost identical to Model 14, except for the robot controller, which has an additional control branch: when the obstacle has already passed the intersection, we want the robot to pass as fast as it can by accelerating fully with maximum acceleration A (no dawdling).

Model 17 Crossing an intersection before a deadline

$$dw_{\text{cxd}} \equiv (ctrl_o; \ ctrl_r; \ dyn)^* \tag{98}$$

$$ctrl_o \equiv ctrl_o \text{ of Model 14} \tag{99}$$

$$ctrl_r \equiv \begin{cases} a := A & \text{if } o > x_o \\ ctrl_r \text{ of Model 14} & \text{otherwise} \end{cases} \tag{100}$$

$$dyn \equiv dyn \text{ of Model 14} \tag{101}$$

Identification of Live Controls Given the robot behavior of Model 17 above, we need to set a deadline that the robot can actually achieve, considering when and how much progress the robot can make while driving (recall that it should still not collide with the obstacle). The deadline has to account for both the robot and the obstacle position relative to the intersection, as well as for how much the robot can accelerate. We start with the easiest case for finding a deadline D: when the obstacle already passed the intersection, the robot simply has to accelerate with maximum acceleration until it itself passes the intersection. The obstacles are assumed to never turn back, so accelerating fully is also a safe choice. The robot might be stopped. So, assuming we start a deadline timer T at time 0, the robot will drive a distance of $\frac{A}{2}D^2$ until the deadline D expires (i. e., until $T = D$). However, since we use a sampling interval of ε in the robot controller, the robot may not notice that the obstacle already passed the intersection for up to time ε, which means it will only accelerate for time $D - \varepsilon$. Formula (102) summarizes this case.

$$\eta^D_{\text{cxd}} \equiv D \geq \varepsilon \wedge x_r - p_x < \frac{A}{2}(D - \varepsilon)^2 \tag{102}$$

If unlucky, the robot determines that it cannot pass safely in front of the obstacle and will have to wait until the obstacle passed the intersection. Hence, within the deadline we have to account for the additional time that the obstacle may need at most to pass the intersection. We could increase D with the appropriate additional time and still start the timer at $T = 0$, if we were to rephrase the implicit definition of the deadline $x_r - p < \frac{A}{2}(D - \varepsilon)^2$ in (102) to its explicit form. In (103), instead, we start the deadline timer with time [k] $T \leq 0$, such that it becomes $T = 0$ when the obstacle is located at the intersection.

$$\eta^T_{\text{cxd}} \equiv T = \min\left(0, \frac{o - x_o}{V_{min}}\right) \tag{103}$$

Verification Theorem 15 uses the deadline conditions (102) and (103) in a liveness property for Model 17.

Theorem 15. Cross intersection before deadline. *Model dw_{cxd} has a run where the robot can drive past the intersection ($p > x_r$). For appropriate deadline choices, all runs of model dw_{cxd}, such that when the deadline timer is expired ($T \geq D$) the robot is past the intersection ($p > x_r$). All runs prevent collision, i.e., robot and obstacle never occupy the intersection at the same time ($p = x_r \rightarrow o \neq x_o$).*

$$\begin{aligned} \phi_{cxd} \wedge \eta^D_{cxd} \wedge \eta^T_{cxd} \rightarrow & \langle dw_{cxd} \rangle (p > x_r) \\ & \wedge [dw_{cxd}]\big((T \geq D \rightarrow p > x_r)\big) \\ & \wedge (p = x_r \rightarrow o \neq x_o) \end{aligned}$$

Proof. We proved Theorem 15 with KeYmaera X. Collision avoidance $[dw_{\text{cxd}}](p = x_r \rightarrow o \neq x_o)$ and liveness $\langle dw_{\text{cxd}} \rangle p > x_r$ follow the approach in Theorem 12. The loop invariant used for proving that the robot always meets the deadline ensures that there is sufficient time remaining until the deadline expires. Similar to the liveness proof in Theorem 12, the deadline is split into two phases, because the robot may not be able to pass safely in front of the obstacle, so it may need to let the obstacle pass first. Recall that $T \leq 0$ when the obstacle is not yet past the intersection, so we characterize the worst-case remaining time until the obstacle passed with minimum speed V_{min} by $T \leq \frac{o - x_o}{V_{\text{min}}}$.

[k] Recall $o \leq x_o$ holds when the obstacle did not yet pass the intersection.

In case the obstacle is not yet past the intersection, the robot must be positioned such that it can pass in $D-\varepsilon$ time, so $T \leq 0 \wedge p + \frac{A}{2}(D-\varepsilon)^2 > x_r$. Finally, once the obstacle passed, the robot has $D-T$ time left to pass itself, which is summarized in $T > 0 \wedge p + s\max(0, D-T) + \frac{A}{2}\max(0, D-T)^2 > x_r$. □

Interpretation of Verification Results

As part of the verification activity, we identified crucial safety constraints that have to be satisfied in order to choose a new curve or accelerate safely. These constraints are entirely symbolic and summarized in Table 6. Next, we analyze the constraints for common values of acceleration force, braking force, control cycle time, and obstacle distance (i. e., door width, corridor width).

Safe Distances and Velocities

Static safety Recall safety constraint (13) from Model 2, which is justified by Theorem 1 to correctly capture when it is safe to accelerate in the presence of stationary obstacles o.

$$\|p-o\|_\infty > \frac{s^2}{2b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon s\right) \qquad (13^*)$$

The constraint links the current velocity s and the distance to the nearest obstacle through the design parameters A (maximum acceleration), b (maximum braking force), and ε (maximal controller cycle time). Table 7 lists concrete choices for these parameters and the minimum safety distance identified by (13) in Model 2. All except the third robot configuration (whose movement and acceleration capabilities outperform its reaction time) lead to a reasonable performance in in-door navigation environments. Fig. 12 plots the minimum safety distance that a specific robot configuration requires in order to avoid stationary obstacles, obtained from (13) by instantiating the parameters A, b, ε and the current velocity s.

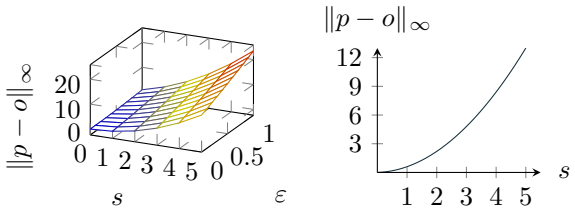
Table 7. Static safety: minimum safe distance and maximum velocity for select configurations

(a) Minimum safe distance

$s\left[\frac{m}{s}\right]$	$A\left[\frac{m}{s^2}\right]$	$b\left[\frac{m}{s^2}\right]$	$\varepsilon\,[s]$	$\|\mathbf{p}-\mathbf{o}\|\,[m]$
1	1	1	0.05	**0.61**
0.5	0.5	0.5	0.025	**0.28**
2	2	2	0.1	**1.42**
1	1	2	0.05	**0.33**
1	2	1	0.05	**0.66**

(b) Maximum velocity through corridors and doors

	$A\left[\frac{m}{s^2}\right]$	$b\left[\frac{m}{s^2}\right]$	$\varepsilon\,[s]$	$\mathbf{s}\left[\frac{m}{s}\right]$
Corridor $\|p-o\| = 1.25m$	1	1	0.05	**1.48**
	0.5	0.5	0.025	**1.09**
	2	2	0.1	**1.85**
	1	2	0.05	**2.08**
	2	1	0.05	**1.43**
Door $\|p-o\| = 0.25m$	1	1	0.05	**0.61**
	0.5	0.5	0.025	**0.47**
	2	2	0.1	**0.63**
	1	2	0.05	**0.85**
	2	1	0.05	**0.56**



(a) By velocity, control cycle **(b)** By velocity

Figure 12. Safety distance for static safety

Table 7b turns the question around and lists concrete choices for these parameters and the resulting maximum safe velocity of the robot that (13) identifies.

Table 8. Passive safety: minimum safe distance and maximum velocity for select configurations

(a) Minimum safe distance

$s\left[\frac{m}{s}\right]$	$A\left[\frac{m}{s^2}\right]$	$b\left[\frac{m}{s^2}\right]$	$V\left[\frac{m}{s}\right]$	$\varepsilon\,[s]$	$\|\mathbf{p}-\mathbf{o}\|\,[m]$
1	1	1	1	0.05	**0.61**
0.5	0.5	0.5	0.5	0.025	**0.28**
2	2	2	2	0.1	**1.42**
1	1	2	1	0.05	**0.33**
1	2	1	2	0.05	**0.66**

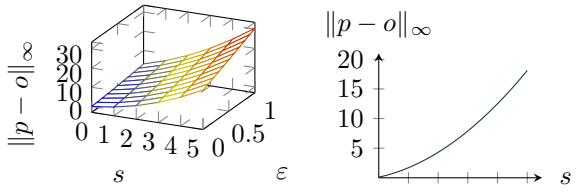
(b) Maximum velocity through corridors and doors

	$A\left[\frac{m}{s^2}\right]$	$b\left[\frac{m}{s^2}\right]$	$V\left[\frac{m}{s}\right]$	$\varepsilon\,[s]$	$\mathbf{s}\left[\frac{m}{s}\right]$
Corridor $\|p-o\| = 1.25m$	1	1	1	0.05	**0.77**
	0.5	0.5	0.5	0.025	**0.69**
	2	2	2	0.1	**0.61**
	1	2	1	0.05	**0.4**
	2	1	2	0.05	**1.3**
Door $\|p-o\| = 0.25m$	1	1	1	0.05	**0.12**
	0.5	0.5	0.5	0.025	**0.18**
	2	2	2	0.1	**0**
	1	2	1	0.05	**0.26**
	2	1	2	0.05	**1**

Moving obstacles Below, we repeat the control constraint (23) from Model 3 for accelerating or choosing a new curve in the presence of movable obstacles. The constraint introduces a new parameter V for the maximum velocity of obstacles.

$$\|p-o\|_\infty > \frac{s^2}{2b} + V\frac{s}{b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(s+V)\right) \qquad (23^*)$$

Fig. 13 plots the minimum safety distance that the robot needs in order to maintain passive safety in the presence of moving obstacles. The maximum velocity in presence of movable obstacles can drop to zero when the obstacles move too fast, the controller cycle time or the maximum acceleration force are too large, or when the maximum available braking force is too small.

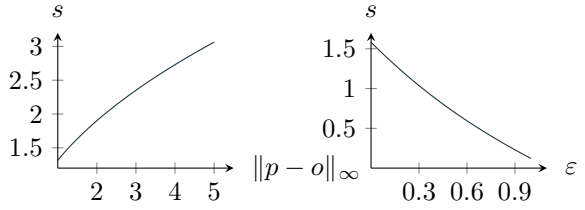


(a) By velocity and control cycle

(b) By velocity

Figure 13. Safety distance for passive safety

Fig. 14 compares the maximum velocity that the robot can travel in order to avoid stationary vs. moving obstacles. The maximum velocity is obtained from (13) and from (26) by instantiating the parameters A, b, ε and the distance to the nearest obstacle $\|p-o\|$. This way of reading the safety constraints (13) and (26) makes it possible to adapt the maximal desired velocity of the robot safely based on the current spatial relationships.



(a) Static safety: by distance for $\varepsilon = 0.05$

(b) Static safety: by control cycle time for $\|p-o\|_\infty = 1$

(c) Passive safety: by distance for $\varepsilon = 0.05$

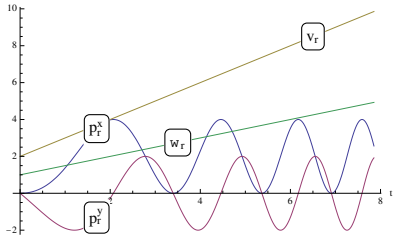
(d) Passive safety: by control cycle time for $\|p-o\|_\infty = 1$

Figure 14. Comparison of safe velocities for static safety and passive safety with acceleration $A = 1$ and braking $b = 1$

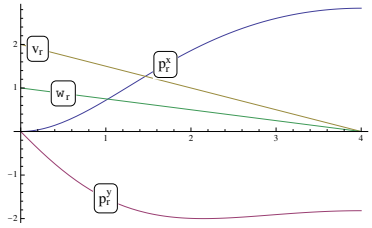
Circular and Spiral Motion

Fig. 15a depicts the position and velocity changes of a robot accelerating on a circle around a center point $c = (2, 0)$. The robot starts at $p = (0, 0)$ as initial position, with $s = 2$ as initial translational velocity and $\omega = 1$ as initial rotational velocity; Fig. 15d shows the resulting circular trajectory. Fig. 15b and Fig. 15e show the resulting curve when braking (the robot brakes along the curve and comes to a complete stop before completing the circle). If the rotational velocity is constant ($\omega' = 0$), the robot drives an Archimedean spiral with the translational and rotational accelerations controlling the spiral's separation distance (a/ω^2). The corresponding trajectories are shown in Figures 15c and 15f.

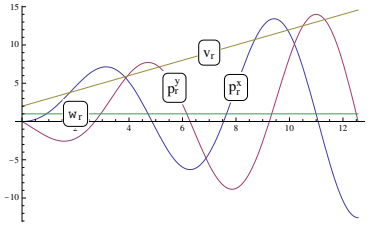
Proofs for dynamics with spinning ($r = 0,\ \omega \neq 0$) and Archimedean spirals ($\omega' = 0,\ a \neq 0$) are available with KeYmaera, but we do not discuss them here.



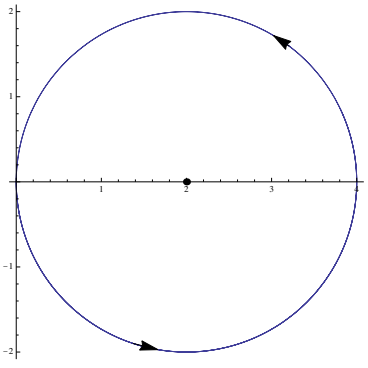
(a) Position (p_r^x, p_r^y), translational velocity s and rotational velocity ω for positive acceleration on a circle.



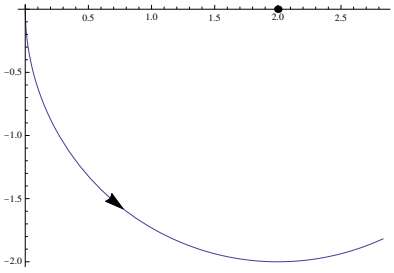
(b) Position (p_r^x, p_r^y), translational velocity s and rotational velocity ω for braking to a complete stop on a circle.



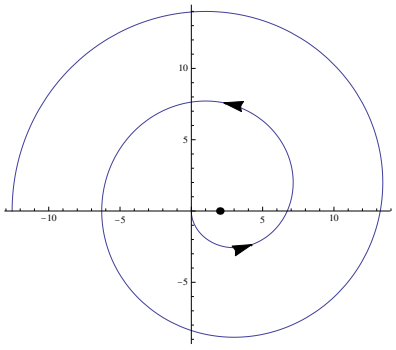
(c) Position (p_r^x, p_r^y), translational velocity s and rotational velocity ω for translational acceleration on a spiral.



(d) (p_x, p_y) motion plot for acceleration a.



(e) (p_x, p_y) motion plot for braking b.



(f) (p_x, p_y) motion plot for c.

Figure 15. Trajectories of the robot over time (top) or in planar space (bottom).