Published in *Autonomous Robots*,32(3):303–331, April 2012. DOI: 10.1007/s10514-011-9271-y
The final publication is available at `www.springerlink.com`

Guaranteeing Functional Safety: Design for Provability and Computer-Aided Verification

Holger Täubig · Udo Frese · Christoph Hertzberg · Christoph Lüth · Stefan Mohr · Elena Vorobev · Dennis Walter

Received: 19 January 2011 / Accepted: 3 December 2011/ Published online: 22 December 2011

Abstract When autonomous robots begin to share the human living and working spaces, safety becomes paramount. It is legally required that the safety of such systems is ensured, e. g. by certification according to relevant standards such as IEC 61508. However, such safety considerations are usually not addressed in academic robotics. In this paper we report on one such successful endeavour, which is concerned with designing, implementing, and certifying a collision avoidance safety function for autonomous vehicles and static obstacles. The safety function calculates a safety zone for the vehicle, depending on its current motion, which is as large as required but as small as feasible, thus ensuring safety against collision with static obstacles. We outline the algorithm which was specifically designed with safety in mind, and present our verification methodology which is based on formal proof and verification using the theorem prover *Isabelle*. The implementation and our methodology have been certified for use in applications up to SIL 3 of IEC 61508 by a certification authority (TÜV Süd Rail GmbH, Germany). Throughout, issues we recognised as being important for a successful application of formal methods in robotics are highlighted. Moreover, we argue that formal analysis deepens the understanding of the algorithm, and hence is valuable even outside the safety context.

Research supported by the German Ministry for Research and Technology (BMBF) under grants no. 01 IM F02 A and 01IS09044B and Deutsche Forschungsgemeinschaft (DFG) under grant FR 2620/1-1.

Holger Täubig, Christoph Lüth, Dennis Walter
Cyber-Physical Systems, German Research Center for Artificial Intelligence (DFKI), Bremen, Germany.
E-mail: {holger.taeubig,christoph.lueth,dennis.walter}@dfki.de

Udo Frese, Christoph Hertzberg, Elena Vorobev
FB 3 — Computer Science, University of Bremen, Bremen, Germany.
E-mail: {ufrese,chtz,elenav}@informatik.uni-bremen.de

Stefan Mohr
Leuze electronic, Fürstenfeldbruck, Germany.
E-mail: stefan.mohr@leuze.de

1 Introduction

Without doubt a grand vision of robotics is the domestic household robot, which assists everyone, but in particular elderly people living independently in their own homes as long as possible (Gates, 2007). Other visions with a shorter road to realisation include the industrial robot co-worker and robots for supporting health-care personnel. All these applications have in common that the robot is operating in a shared workspace with humans. Here safety becomes a central issue.

Small robots, for example vacuum cleaners, can be built inherently safe, i. e. they cannot develop enough force to hurt someone. For a multi-purpose domestic robot this appears unrealistic, and even more so for robots in industrial environments. Here, safety at work will depend on the correct function of sensors, motors, and in particular the soft-

Fig. 1 The SAMS demonstrator driving a right hand bent and the collision-free safety zone of that movement. If there was any obstacle inside the safety zone the AGV would stop.

ware controlling the robot. When taking robots out of the lab into the domestic or industrial realm, safety issues arise at all levels of the development process which have to be handled. In particular, we have to design systems and algorithms with safety in mind.

1.1 Mind the Gap

In most countries development of potentially hazardous machines is regulated by law, directives and relevant industrial standards, resulting in a very rigid software development process which emphasizes documentation and external review. The safety of such a system must be demonstrated to and certified by an external accredited authority. In contrast, robotics software development in academia emphasizes novel capabilities, using rapid prototyping, heuristics or probabilistic methods. Thus, there is a huge gap in the prevailing development methodologies in safety-critical systems on the one hand, and robotics on the other hand, which will hamper commercial applications of service robotics.

The work presented here is an attempt to bridge this gap, by applying formal verification techniques to a robotics algorithm. We need to build bridges from both ends: on the one hand, the usual techniques for developing safety-critical systems do not carry over unchanged, as robotics is a rich and complex domain, and on the other hand, for robotics applications we may need to restrict functionality to an extent which we can verify, both in principle and in practice. An example of this are obstacles: whereas commonly in industrial applications obstacles are considered to be static, academic research is mostly concerned with moving obstacles (Rabe et al, 2007). We have focused our work on static obstacles, as moving obstacles are a current research area and much more complex and non-deterministic. In particular, realizing people detection and tracking in a way that can be verified and certified appears to be very challenging.

1.2 SAMS

In the project SAMS (Safety Component for Autonomous Mobile Systems)[1] we have developed and implemented an algorithm for collision avoidance of a mobile robot, i. e. an automated guided vehicle (AGV), in accordance with the standard most relevant for robotics software, IEC 61508. Fig. 1 illustrates the safety function with a small AGV as demonstrator. The algorithm computes a safety zone depending on the current translational and rotational velocity; if an obstacle is detected inside that zone the AGV stops before colliding with it. We have specified the algorithm's safety function in higher-order logic and proven with an interactive theorem prover that the implementation satisfies this specification (details Sec. 5.6). We have obtained a certificate allowing to use the implementation in safety-critical systems up to safety integrity level (SIL) 3, the highest level for safety at work and one above the level typically needed for AGVs. The certification was based on the following: first and foremost, the above mentioned computer aided proof of the implementation; second, design documents deriving the implemented formula for the AGV's braking behaviour from physical assumptions; and third, additional tests to cover integer-overflow and floating point precision issues. The certification did not cover any aspects of system integration (such as building an actual robot like the demonstrator of Fig. 1).

[1] `http://www.sams-project.org/`

1.3 Formal Software Verification

Safety is established by *safety requirements*, and the conformance of the system (and in particular the software) with these must be verified by means such as tests, code reviews, or tool-supported static analysis. This task is called *verification*. By *formal verification*, we mean the mathematically rigorous, machine-checked proof of correctness of a program with respect to the safety requirements. While well-known in other areas, it is quite novel in robotics. From a formal verification perspective, robotics algorithms are challenging, because robotics requires a mathematically sophisticated domain modelling. Collision is the major hazard created by a robot and hence safety-critical robotics algorithms usually involve a lot of geometry and some physics modeling the behaviour of the robot. Mathematically, geometry argues about properties of subsets of $\mathbb{R}^2$ or $\mathbb{R}^3$. To capture these concepts formally, expressive logics such as set theory or higher-order logic are required, which allow us to formalise textbook mathematics involving real numbers, sets, and functions. We therefore use a general-purpose theorem prover built on higher-order logic as the main verification tool for our robotics application.

The theorem prover takes over the simple proof steps, allowing the user to concentrate on the hard parts. Further, the architecture of the prover allows for a high confidence in the correctness of the proofs (see Sec. 4.1). Moreover, formally proving correctness deepens the understanding of the algorithm, uncovering for example hidden assumptions or cases which were missed by tests or informal reasoning, because of the level of detail necessary for a formal proof (see the discussion in Sec. 7).

1.4 Scientific Contribution

The main contribution reported in this paper is the design and specification of a collision avoidance algorithm

for an AGV with provable safety at work in mind, the formal verification of most of its implementation in C (details in Sec. 5.6), and obtaining a letter of IEC 61508 (SIL 3) conformance. We evaluate our approach, report on lessons learnt beyond this concrete example, and discuss its limitations.

The paper is structured as follows: Section 2 introduces the setting and sketches our algorithm. Section 3 discusses what is required for obtaining a safety certificate. Section 4 introduces the computer-aided theorem proving environment Isabelle and how it has been used to specify our algorithm and prove its implementation. Section 5 presents the algorithm itself and points out how it is designed for provable safety. Section 6 shows experiments demonstrating that the algorithm is effective and practical, whereas Section 7 evaluates our design and formal software verification process regarding lessons learnt and the certification effort.

2 The SAMS Algorithm

The purpose of the developed algorithm follows the standard definition of safety at work for AGVs: it allows to prevent a vehicle moving in a plane from colliding with static obstacles. More precisely, the algorithm computes a *safety zone* that is a superset of the *braking area*, i.e. the area covered by the vehicle during braking to a standstill from the current velocity. The result can be safeguarded by a laser rangefinder (if an obstacle is inside, the vehicle has to stop, otherwise it can safely continue), although this is not part of the verified algorithm.

Figure 2 introduces the algorithm, its two steps and three stages of development. We will refer to these throughout the paper.

2.1 Input

The algorithm takes as input intervals $[v_{\min}, v_{\max}]$ and $[\omega_{\min}, \omega_{\max}]$ which safely cover the true translational and rotational velocities v and ω of the vehicle. The role of the intervals is to cover measurement uncertainties. As configuration parameter it takes a set of points $[R_i]_{i=1}^{n}$ which define the robot's shape as their convex hull, and a list $(v_1, s_1), \ldots, (v_m, s_m)$ of braking measurements for straight movements which define the vehicle's braking behaviour. Each pair consists of a velocity v_j ($\omega = 0$) and the corresponding measured distance s_j the vehicle needed for braking. One measurement must be taken at maximum speed. Furthermore, a latency $\triangle t$ is given which specifies the time the vehicle continues to drive with velocity (v, ω) before it starts to brake; it comprises the sum of the program's cycle time as well as any latency in the input data and the reaction time of the brakes.

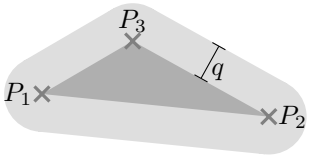


Fig. 3 Sphere Swept Convex Hull representation used inside the algorithm. The representation consists of an unsorted array of points $[P_k]_{k=1}^{K}$ and a buffer radius q. The area expands the convex hull $\mathrm{conv}\left\{[P_k]_{k=1}^{K}\right\}$ of the finite set of points by the buffer radius. That means, it contains all points having distance of at most q to any point of the convex hull (obviously including all points of the convex hull). Actually, the algorithm works in 2D and the sphere is a circle, but we keep the "sphere swept-" prefix because it is an established textbook convention (Ericson, 2005, Ch. 4.5). We further proposed a 3D extension for collision detection of robot manipulators in Täubig et al (2011).

2.2 Output and Guarantees

The algorithm computes the safety zone as a subset $\subset \mathbb{R}^2$. The subset is represented as a so-called Sphere Swept Convex Hull (SSCH, Fig. 3) by the algorithm.

The algorithm guarantees that for any velocity $v \in [v_{\min}, v_{\max}]$ and any rotational velocity $\omega \in [\omega_{\min}, \omega_{\max}]$ no part of the vehicle will leave the safety zone at any time while first driving with constant velocity $(v, \omega)^T$ for time $\triangle t$ and then braking down to standstill according to the braking model. By including the cycle time in $\triangle t$ it is ensured, that no obstacle will be hit although the obstacle has to violate the safety zone before an emergency stop is triggered.

This guarantee is established using a combination of computer-aided and pen-and-paper proofs (details Sec. 5.6). In particular, the core algorithm, i.e. implementation of step 2 and the algorithm (49) defining step 2*, are verified formally.

2.3 Physical Assumptions

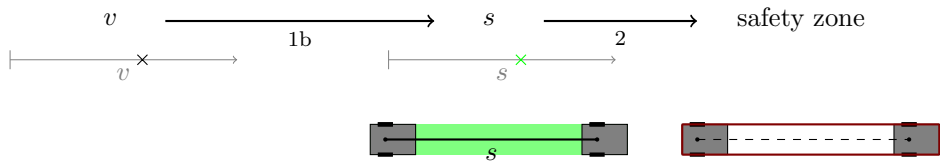
The algorithm and its certification are based on the following physical assumptions.

(A1) *Static obstacles*: It is sufficient to consider the current position of obstacles without extrapolating their movement.

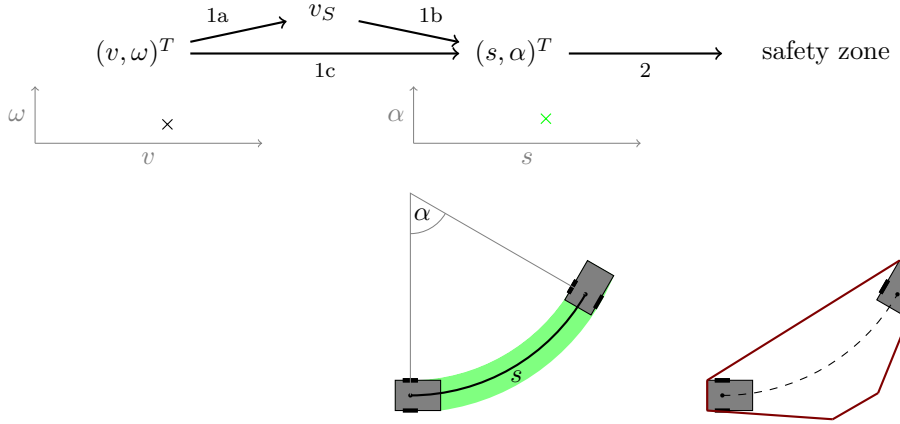
(A2) *Circular braking*: The vehicle keeps the same curvature (steering angle, $\frac{v}{\omega}$) while braking, so the braking trajectory is a circle (resp. straight line for $\omega = 0$).

(A3) *Energy dissipation*: The rate $|\dot{E}|$ of dissipation of kinetic energy E in the brakes is a) the same for all $(v, \omega)^T$ motion-states with the same kinetic energy E; b) increases at most proportional to v^2 (i.e. $|\dot{E}(v)|/v^2$ does not increase); c) does not decrease with growing v; d) is independent of location and time.

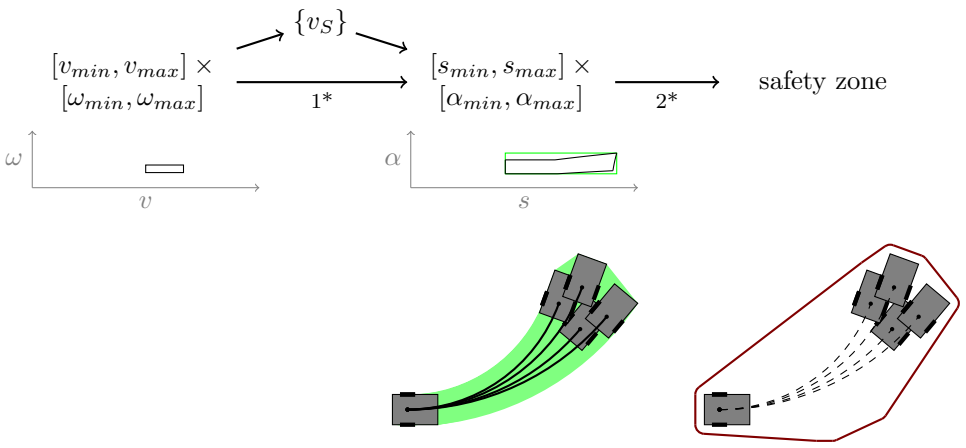
We have only listed (A1) to be explicit, but it is not technically needed by the algorithm. This is because the algo-



(a) The first development stage considers only straight motion: Step 1b starts from the measured velocity v and computes the braking distance s needed by the vehicle to stop. The quotient s/v is called characteristic time and needed later. The illustration shows the straight trajectory of the vehicle. Step 2 computes the safety zone (red polygon, right) from s as the area touched by the robot during braking (green area, center) or a superset thereof.



(b) The second development stage extends to curved motion: Step 1 takes velocity v and angular velocity ω as input and computes the resulting 2D braking distance. It is expressed as a distance s along a circle and an angle α based on the assumption that the vehicle stays on the same circle during braking (see center illustration). The algorithm first computes an equivalent straight speed v_S with (at least) the same kinetic energy as $(v,\omega)^T$ (1a). Then it computes the characteristic time from a straight motion with v_S as in the top row. It assumes that this time is the same for the energetically equivalent $(v,\omega)^T$ (cf. discussion of assumption (A3) later). From this time and the initial curvature v/ω the algorithm computes s and α (1c). As $v_S > v$ this procedure takes the rotational kinetic energy of the vehicle conservatively into account and is even valid for rotating on the spot. Step 2 computes the safety zone (red polygon, right) in our internal representation a subset of $\mathbb{R}^2$ covering the area touched by the robot (green area, center). For the 2D motion this is of course more complex and the main part of the algorithm.



(c) The third development stage incorporates uncertainties in the measured $(v,\omega)^T$ and is the final algorithm proposed in this paper. This extension is denoted by an asterix in the step number. Step 1* takes an interval $[v_{\min}, v_{\max}] \times [\omega_{\min}, \omega_{\max}]$ instead of a single $(v,\omega)^T$ and maps it to an interval $[s_{\min}, s_{\max}] \times [\alpha_{\min}, \alpha_{\max}]$. As the mapping is non-linear, the computed interval (green rectangle, center diagram) is not exact but conservatively bounds the complex shaped exact result in $(s,\alpha)^T$ space (inner black line, center diagram). The center illustration shows the union of all possible braking trajectories with the four extremal cases highlighted. Step 2* computes the safety zone from $[s_{\min}, s_{\max}] \times [\alpha_{\min}, \alpha_{\max}]$ by applying the algorithm of the middle row to the four extremal combinations and extending the result by a bound for additional nonlinear effects that occur. The latter extension is visible as a gap between the vehicle stop-positions and border of the computed safety zone (right).

Fig. 2 Overview of the proposed algorithm. The arrows show the two main steps (1-2), their input and output. The rows (a)-(c) correspond to different development stages of the algorithm introduced to facilitate understanding the idea behind it.

brake/friction type	deceleration	energy dissipation	allowed
power limited	$\propto v^{-1}$	const	yes
Coulomb	const	$\propto v$	yes
proportional	$\propto v$	$\propto v^2$	yes
viscous	$\propto v^2$	$\propto v^3$	no

Fig. 4 Types of friction respectively brakes allowed by (A3b-c). The symbol $\propto$ means proportional. The deceleration and dissipated energy depends on the velocity v. (A3b) specifies an upper bound and limits how fast both can grow with v. In particular, it rules out viscous braking (e.g. a parachute). (A3c) is a trivial lower bound. stating that the energy dissipation may not decrease with increasing velocity.

rithm just computes a safety zone reflecting what the vehicle does and the assumption only comes in later when comparing the computed safety zone with data from a laser rangefinder. Further, if we knew motion and shape of an obstacle we could compute an obstacle safety zone and intersect both safety zones to determine whether braking must be triggered. However, this is beyond the scope of this paper.

Assumptions (A2) and (A3) are physical assumptions on the vehicle's behavior. The certification authority has accepted them as adequate for industrial environments. (A2) and (A3a) are approximations, while (A3b-c) are upper and lower bounds, thus strictly conservative. Practically, (A3b-c) restrict the type of friction resp. of brakes to which our calculation of the braking distance applies (Fig. 4). They are used in a pen-and-paper proof (Sec. 5.2.1-5.3.2) to derive the braking model, i.e. a formula for the braking distance $(s, \alpha)^T$ as a function of $(v, \omega)^T$. As a whole, this is step 1 in Fig. 2b.

In detail, (A2) justifies the model of the braking motion as a circle described by $(s, \alpha)^T$; (A3a) allows to bound the so-called characteristic time of circular braking from the straight braking measurements (step 1a); (A3b) allows to interpolate the braking measurements with regard to v (step 1b); and (A3c) allows to extrapolate (slightly) beyond the maximum measurement (step 1b). (A2) is also needed to convert the characteristic time back to a circular braking distance (step 1c). The role of the characteristic time here is being a single-number parametrization of "how long it takes to brake" that is applicable to straight-forward motion, circular motion, and turning on the spot. In the latter case the conventional "braking distance", i.e. the arc-length of the vehicles reference point, becomes zero and cannot be used as a parametrization. However, we call it characteristic time as it has characteristics of time but it is not the real physical time until stand still (cf. (23) and (24) in Sec. 5.3.2). Assumption (A3a) is not strictly conservative as it abstracts from the different wheels to a single point braking force, but it is fairly exact and robust under physical considerations (see Sec. 6.3). At least, it is more conservative than ignoring the rotation velocity ω completely, which is quite common in industry.

2.4 Assumptions Defining the System Scope

(A4) *Correct verification*: The verification tools, i.e. our verification environment and the theorem prover, operate as specified, and in particular never prove false statements. The implementation data types such as `int` and `float` can be abstracted by $\mathbb{Z}$ and $\mathbb{R}$.

(A5) *Correct system integration*: The implementation is correctly integrated into the target platform, and the target hardware and compiler work as specified.

(A6) *Real-time computation*: The execution time of the algorithm fits into the available cycle time of the target platform.

(A4) was reviewed by the certification authority and the finite precision and overflow issues tested (see the discussion in Sec. 4).

The formally verified software covers the computation of the safety zone. Overall system safety additionally requires correct system integration. Assumptions (A5) and (A6) reflect issues out of the scope of the verification environment that are delegated to the integrator of the target platform. These assumptions were explicitly listed in a *user manual*, which was part of the reviewing process as well. Basically, the system integrator has to follow industrial state-of-the-art.

For (A5), the integrator is responsible that the API of the SAMS component is used as specified, in particular that the input and configuration is correct and that the results are acted upon appropriately. The integrator further has to ensure that the target and development platform satisfy the safety integrity level (see Sec. 3.1) of the target system, in particular that the compiler used is certified or proven in use. Regarding our own compilation process, we prohibit optimization and do not make use of the full C language, but a (subset of) MISRA C, an industry-proven restriction of C particularly aimed at safety-critical systems.

For (A6), although the verification cannot make any runtime guarantees the runtime has a reproducible upper bound because of the absence of dynamic memory allocations and can thus be handled with integration tests on the target platform.

2.5 Assumptions Regarding the Treatment of Uncertainty

The robustness of the collision detection is based upon the idea of error bounds to make the physical assumptions and measurement interpretation realistic. This goes for both the input intervals as well as measurements outside the scope of the verified software.

(A7) *Correct input intervals*: 100% of the readings that quantify v and ω are within a given error bound;

thus their true values lie within $[v_{\min}, v_{\max}]$ and $[\omega_{\min}, \omega_{\max}]$.

(A8) *Free space*: After a state-of-the-art outlier handling all readings of the laser rangefinder are within a given error bound; thus subtracting the error bound yields a guaranteed lower bound of the free space.

The formal verification, i. e. (A7), assumes the input interval to be correct; its computation based on an appropriate error bound is part of the system integration. This error margin is commonly identified during the certification process on the system level, e. g. as a quantile of an error distribution, and thus out of the scope of this paper. Formally, (A7) assumes that all readings of the sensor are within the error bound. However, intervals allow to determine a fixed failure rate according to the norm, if that becomes necessary. Besides being easy to use the intervals further compensate for the model error induced by (A2), i.e., enlarging $[v_{\min}, v_{\max}]$ and $[\omega_{\min}, \omega_{\max}]$ also makes the safety zone contain deviations from the perfect circular motion.

(A8) is out of the scope of the verified system and just requires to reuse existing and already certified technology inside current safety laser rangefinders. These contain algorithms for safeguarding a given safety zone, in particular outlier handling.

To summarize, the assumptions (A1) – (A8) are field-proven in industry, realistic, and have been reviewed and accepted by the certification authority concerning the use in industrial environments. Some may seem to be below scientific state-of-the-art but we rather view them as an improvement of the industrial state-of-the-art.

2.6 Practical Advantages of the Algorithm

Apart form being strictly conservative with mathematical rigour, our algorithm has practical advantages:

(i) The safety zone is not preconfigured as is common in industry, but is computed in real-time. Even small changes in velocity result in perfectly adjusted safety zones.
(ii) The vehicle's shape can be an arbitrary convex polygon.
(iii) By explicitly computing a safety zone the algorithm separates vehicle aspects from sensor aspects of collision avoidance.
(iv) With only a single measurement procedure of braking from straight motion the algorithm safeguards arbitrary translational and even rotational movements.

The last point is easily underestimated from an academic perspective. In industrial practice, measuring the rotational braking behaviour complicates the setup procedure a lot and is a serious obstacle to applicability. So it is very helpful that the algorithm needs only one straight braking measurement to conservatively bound braking from curved or purely rotational motion. This bound is derived from (A2) and (A3) in Sec. 5.3 in a pen and paper proof. The result is verified in simulation in Sec. 6.3.

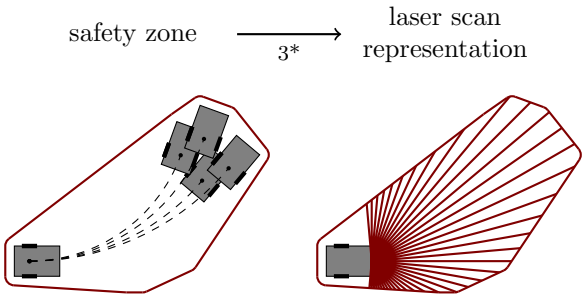


Fig. 5 Postprocessing step 3: Convert safety zone from internal Sphere Swept Convex Hull (SSCH) representation into a laser scan like representation. In this representation the safety zone can be safeguarded by simply comparing with a laser scan.

2.7 Postprocessing

In our internal SSCH representation it is difficult to compare the safety zone with a laser scan. To facilitate this there is a postprocessing step 3* (Fig. 5) that converts the safety zone into a laser scan. In this representation, each ray the distance stored in the safety zone is simply compared with the distance measured by the laser rangefinder.

2.8 Related Work

Collision avoidance has been a subject of research from the early days of mobile robotics on (Nilsson, 1984). It has usually been viewed more generally as obstacle avoidance, i. e. finding a way around obstacles.

Regarding certified collision avoidance, the current state-of-the-art is to have a safety laser rangefinder but no algorithm at all. Instead, only fixed preconfigured safety zones for different turns and velocities are manually defined and monitored by the laser rangefinder. For AGVs test procedures are described and required in EN 1525.

The most well known academic obstacle avoidance method is the dynamic window approach (DWA) by Fox et al (1997). It searches through different combinations of $(v, \omega)^T$ in the window defined by acceleration limits, i. e. the dynamic window. It considers only those combinations where the robot can stop along the circular trajectory before hitting the closest obstacle. Among those it chooses the combination leading most directly to the goal. The core operation from the perspective of collision avoidance is to determine whether the robot can stop on a given circle before an obstacle. Fox et al. assume a circular robot, so the

distance to collision along the circular trajectory can be analytically computed for each obstacle. This is done for all obstacles and the minimum taken. The DWA is extensively used in tour guide robots and has later been extended by Philippsen and Siegwart (2003) to include a local planning component. Compared to our algorithm, the disadvantage of DWA is the limitation to circular robots.

Lankenau and Röfer (2001) propose a method to handle arbitrarily shaped robots by precomputing the safety zone for every $(v, \omega)^T$. Their algorithm is however too slow to compute a safety zone in real-time, so their method can not be used when parameters or the vehicle shape changes, for instance due to load. Furthermore, since they use a grid approximation, it appears unlikely that the algorithm is actually correct in a strict mathematical sense.

Schlegel (1998) also uses precomputed tables. These tables store for every curvature in a grid around the robot in every cell how fast the robot would be allowed to go if there was an obstacle in that cell. This is very similar as the table used by Lankenau and Röfer, but indexed by the obstacle coordinate, not by the vehicle speed. It shares the same disadvantage, which all methods with precomputation have.

A different class of algorithms are the potential field methods introduced by Khatib (1986), that calculate a virtual repulsive force from nearby obstacles on the vehicle. To our knowledge these methods do not give strict mathematical guarantees with realistic braking assumptions. Victorino et al (2003) choose a similar approach by defining control laws that let the vehicle move along Voronoi edges of the sensor-perceived environment. This implicitly avoids collision, however again does not model braking capabilities.

The nearness diagram (NDD), developed by Minguez and Montano (2004), is a clever set of heuristics for different obstacle situations. It has been used on a wheelchair which is significantly non-circular. Other algorithms, such as DWA and the ones by Röfer and by Schlegel, model the braking behavior and make geometric computation based on this model and the vehicle shape. So their construction suggests a correctness proof, even if minor details, such as grid approximations probably need to be made conservative. By contrast, NDD does not consider deceleration bounds explicitly, so a formal proof considering a vehicle with limited acceleration appears much harder.

Fraichard (2007) introduces three safety criteria that should be addressed from a motion safety point of view. These are the dynamics of the robot and people (or objects) in the environment, and the consideration of an unlimited time horizon. Parthasarathi and Fraichard (2007) claim to satisfy these criteria. They propose an inevitable collision state-checker based on the general method by Fraichard and Asama (2004). The algorithm avoids inevitable collision states, which are robot states where every control input will lead to a collision. It assumes known velocities and shapes of all obstacles. In principle this can be provided, but providing it with the integrity needed for a safety certification appears to be far ahead. Our focus is different: On the one hand, we use formal proofs instead of pen-and-paper; on the other hand, we do not aim as high because we limit the information to the dynamics of the vehicle resulting in guarantees that are somehow more local, e. g. the vehicle does not leave the computed safety zone, but combined with industrial state-of-the-art assumptions such as static obstacles it yields an applicable safety device in conformance with current safety standards.

An earlier method also assuming knowledge of the dynamics of the environment is the velocity obstacles method by Fiorini and Shillert (1998). The set of robot velocities which would collide with obstacles moving at a known velocity are calculated and a feasible velocity outside this set is chosen.

Collision avoidance and car verification is also a topic in the automobile industry. Again, there is the industrial state of the art on the one hand, and more visionary academic work on the other hand. Commercially available driver assistants that warn the driver or even initiate braking work on the basis of radar and/or computer vision. An overview of this complex technology is given by Winner et al (2009). These systems are governed by a special norm ISO 15623 (2002) and often developed according to ISO 26262 (2011) for functional safety in automobile systems which is a specific automotive replacement for IEC 61508 used in our work. Other, research-oriented work considers distributed cooperative control. These approaches model the behaviour of car movements on a road network, and show that if the car movements satisfy certain criteria (e. g. driving in platoons (Varaiya, 1993; Dolginova and Lynch, 1997; Puri and Varaiya, 1995) or using adaptive cruise control (Loos et al, 2011)) no collisions can occur. They assume a global model and global knowledge such as the vehicle position, whereas our focus is more local using available sensor data. Because they model an idealised version of the global system, they can derive global safety properties, but of the idealised model. In contrast, we can derive local safety properties covering real-world aspects such as measurement uncertainties. Other approaches to collision avoidance for autonomous vehicles include generating safe paths (e. g. by Du Toit et al (2008) for the Alice autonomous car), and a verified planner-controller subsystem by Wongpiromsarn et al (2009), which reliably follows a generated path. Althoff et al (2010) also consider car collision avoidance, but focus on evasive manoeuvres instead of braking.

To sum up, our algorithm differs from algorithms found in the literature by its mathematical rigour, which is motivated by our approach to use formal software verification for the official certification. On the other hand it only stops the vehicle but does not negotiate the obstacle.

3 Standards-Compliant Software Development

The development of potentially hazardous software has by law to follow the relevant industrial standards. In this section we briefly review standards applicable to robotics software, and describe how it impinges our development process.

3.1 Applicable Standards

Automated guided vehicles (AGV) are machinery according to the definition of the European machinery directive (European Parliament and Council, 2006) and hence (in the EU) its provisions apply to them. The directive's primary goal is to state and enforce health and safety requirements relating to the design, construction, and placing on the market of machinery. Part of these provisions is the carrying out of a risk assessment "*to determine the health and safety requirements which apply to the machinery*". Due to its wide application area the directive is necessarily too general to be used directly for proving compliance with the requirements stated therein. Compliance is instead proven by the proper application of domain-specific standards. In the area of mobile robotics, the standardisation of safety requirements is still somewhat underdeveloped, although there are several regulations that apply to stationary robots (e. g., ISO 10218-1 and ANSI R15.06). Hence, for mobile robots the applicable standard is the more general IEC 61508, '*Functional safety of electrical/electronic/programmable electronic safety-related systems*', which is applicable to all kinds of programmable systems for which no more specific standard exists.

IEC 61508 states requirements for each phase of the system life cycle. It suggests a development process which is a variant of the V-model, and requires a variety of documents to be produced. These include process-oriented documents such as software quality management plan or a verification and validation plan, and product-oriented documents.

A central concept introduced by the standard is the *safety integrity level* (SIL). Four levels of safety integrity are defined, ranging from low integrity (SIL 1) to high integrity (SIL 4). The SIL represents the probability that a safety function will operate according to its specification. The *target* SIL assigned to a safety-related system essentially dictates which safety requirements it must satisfy, and how demanding these are. Collision avoidance for AGVs is typically assigned SIL 2, as (equivalently) stated in EN 1525. Measures to establish or increase safety at work are typically SIL 3 applications, e. g., access control for hazardous work places via light curtains. In the railway domain one predominantly encounters SIL 4 applications such as safety functions concerned with railway signal controlling.

3.2 Applying IEC 61508

Conformance of our implementation with the requirements of IEC 61508 was certified by an external, accredited certification authority[2]. We followed the development process suggested by the standard, emphasizing verification by theorem proving (i. e., *formal verification*). The development starts with concept papers describing the algorithms, and a hazard analysis (in our case a failure modes and effects analysis, FMEA). The latter lead to safety requirements, which are broken down from global requirements referring to the overall system to local requirements relating to individual modules. In our case, while the global requirements were formulated in prose, the local safety requirement specifications were formal annotations of the relevant C functions (see Sec. 4).

Showing that the implementation conforms with the specifications is called verification. For the life cycle phases of module design and coding we used the following five verification methods:

- code reviews,
- formal specification of the program functions,
- reviews of the formal specifications,
- formal proof as described in Sec. 4,
- and dynamic analysis and test.

From this, code review is a standard technique, but in our case the code reviews were not the main method of verification, but rather lead up to the requirement of the formal specifications. The major technique for verifying the functional requirements of a software system is dynamic analysis, i. e. testing. This is where most of the time in a common verification effort is spent. In our case, formal verification largely replaced functional testing as it ensures functional correctness. The only tests that had to be performed on the module level were related to over-/underflow and numerical stability (Sec. 4.6). No functional testing had to be performed for the formally verified units, due to the level of detail of our specifications and the rigour implied by formal verification.

For the implementation (coding phase), we used a subset of MISRA C (MISRA, 2004), itself a subset of C for safety critical applications; the standard 'highly recommends' the use of such subsets for the C language in safety-critical applications. A detailed comparison of how our verification methods cover the measures required in the standard can be found in the appendix (Sec. A).

3.3 Letter of Conformance

All activities were audited by the certification authority. Conceptual documents were reviewed throughout the speci-

[2] TÜV Süd Rail GmbH, `http://www.tuev-sued.de/rail`

fication process. These were (safety) requirement specifications, concept papers for the braking model (step 1*), and the safety zone computation (steps 2* and 3*), as well as the verification and validation plan. Additionally, an on-site audit was done by the certification authority in which the verification of several example programs as well as concrete functions from the SAMS software were demonstrated.

Based on the on-site audits and the reviews of the verification and validation plan in particular, the certification authority issued a letter of conformance, asserting "*[...] that SAMS software, the software for computing velocity-dependent safety zone for autonomous mobile robots, has been developed according to the software development process as laid out in IEC 61508-3:2008 (SIL 3). The related analyses and tests have shown that there are no safety related objections against the use of SAMS software for the computation of velocity dependent safety zone.*" Formal proof was recognised as the central means of verification.

4 Formal Verification and Proof

Our formal verification focuses on *functional correctness*, comprising the absence of runtime errors such as array-out-of-bounds or division-by-zero, and correctness of the results of computations as defined by formal specifications. Functional correctness is mandated by IEC 61508 and other standards to ensure program safety properties on the code level, and moreover we consider it important in the context of robotics, where algorithms such as the one presented here involve very complex computations whose correct implementation is hard to verify by a code review or tests.

To state and prove safety properties of C code, we need three ingredients: a formalisation of key aspects of the robotics domain (here, the geometry of objects and motion in the plane, e. g. the convex hull of a set, or rigid body transformations); a language which can specify the safety properties of C functions using this formalisation (e. g. this function computes the convex hull of a set of points); and a tool which allows us to prove that C code satisfies the specification as stated.

4.1 Isabelle

Our main verification tool is Isabelle (Nipkow et al, 2002), an interactive theorem prover which has been in development for over twenty years. Isabelle allows the user to state properties and prove them interactively. Proofs are interactive and computer-aided rather than fully automatic: the user writes proof scripts containing intermediate proof steps, the correctness of which is checked by the theorem prover which also offers considerable automatic support by rewriting or tableaux proofs. For example, a typical proof script about simple properties of the natural numbers would consist in two steps, first performing induction on a specific variable, and then solving the cases by automatic rewriting. Larger developments are written as hierarchical proof scripts, containing proofs as well as definitions.

Isabelle is an LCF-style prover (Gordon et al, 1979), i. e. it is based on a small logical core encoding the axioms of higher-order logic[3], and only allows to derive new theorems by sound logical inferences. This reduces the question of correctness of any proof in Isabelle to correctness of that core, which is small and stable enough to check manually. The question of correctness had to be addressed during the certification; Isabelle's architecture together with the fact that it is supported by an active research community, has proper documentation and a large enough number of global usage hours[4] sufficed.

4.2 Domain Modelling

Many definitions and theorems concerning basic geometry can directly be transformed from textbook math to Isabelle. For example, a set is called *convex* if for every pair of points within the set, every point on the line segment that joins them is also inside. In Isabelle:

```
definition segment :: "Point ⇒ Point ⇒ Point set"
where "segment x y =
  {z. ∃t . 0≤t ∧ t≤1 ∧ (z = t *R x + (1-t) *R y) }"

definition is_convex :: "Point set ⇒ bool"
where "is_convex K = (∀x∈K. ∀y∈K. segment x y ⊆ K)"
```

And the convex hull of a set X is the smallest convex set containing X:

```
definition conv :: "Point set ⇒ Point set"
where "conv X = ⋂{K . is_convex K ∧ X⊆K}"
```

Using these definitions many simple theorems can be proven by just inserting definitions and automatic simplifications:

```
lemma conv_monotone: "X ⊆ Y ⟹ conv X ⊆ conv Y"
  by(auto simp add: conv_def)

lemma conv_Union: "conv X ∪ conv Y ⊆ conv (X ∪ Y)"
  by(auto simp add: conv_def)
```

At some points we had to prove certain inequations containing sine and cosine functions. Therefore inequations of the form $\cos x \geq 1 - \frac{x^2}{2}$ were proven by using the mean value

[3] Isabelle is actually generic and can handle e. g. the axioms of ZF set theory as well, but we use higher-order logic here.

[4] We actually estimated the number of hours that Isabelle has been in serious use (as $2 \cdot 10^6$ hrs). This technique of showing that a tool has 'increased confidence from use' is commonly applied for non-certified compilers.

@requires	Precondition
@ensures	Postcondition
	Pre- and postcondition are written in C-like syntax, with Isabelle code embedded by ${...}. The result of the function is referred to as **\result**, and the postcondition may refer to the value of parameter x on entry by **\old**{x}.
	Memory layout constraints are part of the precondition and specify that parameters are valid pointers (**\valid**) or that the memory regions they refer to are disjoint (**\unrelated**, or **\separated** for arrays).
@modifies	Modification frame specification, specifies those references which the function body may modify.

Fig. 6 Elements of function specifications. All elements are optional.

theorem. In fact it was even possible to make general statements such as for $t \geq 0$ and even $n \in \mathbb{N}$:

$$\cos t \leq \sum_{k=0}^{n} \frac{(-1)^k t^{2k}}{(2k)!}, \qquad \sin t \leq \sum_{k=0}^{n} \frac{(-1)^k t^{2k+1}}{(2k+1)!}. \qquad (1)$$

Surprisingly complicated on the other hand was to use these formulae for concrete values of n and automatically simplify equations containing it. Basically this shows that Isabelle is not a computer algebra system.

4.3 Specifying Functional Correctness

To express the functional properties of interest we designed a formal language for the high-level specification of the functional behaviour of C programs. The language annotates functions with specifications comprising preconditions, postconditions, a required memory layout, and a modification frame limiting the effect of function execution on memory changes (see Fig. 6). The specification states that if the precondition holds and the memory layout is as described when calling the function, then the function will terminate, returning a state satisfying the postcondition and the modification frame (i. e. it will only change locations in the state as allowed by the modification frame). Fig. 7 shows the specification of a function that composes rigid-body transforms as an example.

Now we discuss how the safety function intuitively described in Section 2.2 translates to our formal specification language. On purpose, we discuss the specification before the algorithm, because this is the idea behind "programming by contract": From the outside, one only needs to understand what a function does not how the function does it.

As said, step 1 boils down to a formula and we have specified and formally verified that the implementation actually implements this formula. The derivation of the formula from the physical assumptions (A2-3) was done as a pen-and-paper proof in a reviewed design document as part of the certification but not in Isabelle. So we will concentrate here on the most insightful steps 2 and 2* of the algorithm.

```
1  /*@
2    @requires ^is_RT(a2b) && ^is_RT(b2c)
3         && \valid{a2b, b2c, a2c}
4         && \unrelated(a2b, a2c)
5         && \unrelated{b2c, a2c)
6    @ensures ^is_RT(a2c) &&
7      ${ ^RT{a2c} = ^RT{b2c} o ^RT{a2b} }
8    @modifies *a2c
9    @*/
10 void comp_transform(const RigidTransform *a2b,
11                      const RigidTransform *b2c,
12                      RigidTransform *a2c);
```

Fig. 7 An example specification of a C function compounding rigid body transforms. The main statement is found in the postcondition **@ensures** in line 6. It states that the result a2c interpreted as a rigid body transform, i.e. a mapping $\mathbb{R}^2 \mapsto \mathbb{R}^2$, is the composition $\circ$ of a2b and b2c interpreted as rigid body transforms. RT is what we call a representation function, which lifts a C value of type RigidTransform into its Isabelle-domain equivalent. This statement is an Isabelle statement on the domain level, as indicated by the ${...} around it. It is a particular strength of our style of specification to be able to state not only properties of variables on the C-level but also of their abstract meaning on the Isabelle domain level. is_RT is a data type invariant that ensures that C values actually represent rigid body transforms. It is used in the precondition **@requires** (line 2) for the input and in the postcondition (line 5) for the output. The memory layout constraints require that a2c is not aliased with a2b nor with b2c (line 4) and that all three are valid pointers (line 3). The modification frame **@modifies** specifies that the function only changes the location pointed at by a2c.

Fig. 8 explains the specification of the safety zone computation for a single $(s,\alpha)^T$ (step 2). As can be seen, the use of high-order logic allows to specify the safety function concisely, elegantly, close to physical reality and far away from the implementation. In words it means: When the robot brakes according to the input $(s,\alpha)^T$ no part of the robot leaves the safety zone. A proof of such a specification not only shows there is no implementation error. It further gives strong confidence that the function does the *intended* job. It is much stronger than it would be possible for proving a more technical specification, e.g. one limited to computational expressions.

Fig. 9 explains the specification of the safety zone computation for s,α intervals (step 2*), i.e. with incorporation of uncertainty. It can be observed that it is very similar to the single $(s,\alpha)^T$ case, just stating that the safety-property must holds for any $(s,\alpha)^T$ inside the given intervals. This is very directly, what we intend the function to do, namely computing a safety zone that is safe for all possible s,α. It is far from the implementation (see Sec. 5) where the generalization to intervals raises issues which are not there for single $(s,\alpha)^T$. Hence, a proof with this specification gives high confidence that there is no common misconception in specification and implementation and the function actually does the *intended* job.

```
1  /*@
2    @requires \separated(startpoints_data, startpoints_len, result_data, result_len_max)
3           && \unrelated(result_data, &sams_other)
4           && \unrelated(startpoints_data, &sams_other)
5           && 4 < l  && (l + 1) * startpoints_len <= result_len_max
6
7    @modifies sams_other, result_data[:(l + 1) * startpoint_len]
8
9    @ensures \result == sams_safe -->
10     ${ let S = ^Vector2DList{startpoints_data, startpoints_len};
11            R = ^Vector2DList{result_data, (l + 1) * startpoints_len};
12            ROBOT = conv S;
13            SAFETYZONE = conv (S ∪ R)
14        in ∀ p ∈ ROBOT . (arc `s `alpha p) ⊂ SAFETYZONE
15      }
16   @*/
17 SAMSStatus safetyzone_salpha( Float32 s, Float32 alpha, Int32 l,
18                               const Vector2D * startpoint_data, Int32 startpoint_len,
19                               Vector2D * result_data, Int32 result_len_max );
```

Fig. 8 Actual specification source (with Isabelle pretty printing) of the function implementing step 2, i. e. computing the safety zone from $(s,\alpha)^T$. Lines 2–4 specify the usual technical assumptions on pointers and array length. Line 5 requires enough space in the output array. Line 7 specifies the locations which this function may modify. The postcondition starts with the condition that the return value does not indicate an error (line 9). The main statement is in line 14, saying that for every point p of the robot (given by ROBOT) the $(s,\alpha)^T$-arc which p traverses is contained in the computed safety zone (given by SAFETYZONE). ROBOT refers to the convex hull of the list of points in startpoints_data (line 10, 12). SAFETYZONE refers to the convex hull of the computed points in result_data (line 11,13) plus the ones in startpoints_data for technical reasons. As can be seen, Isabelle can reason about infinite sets, such as a convex hull $\subset \mathbb{R}^2$ and quantify over infinite many values which is not possible in a computational environment.

```
1  /*@
2    @requires s_min <= s_max && alpha_min <= alpha_max
3      && \separated(startpoints_data, startpoints_len, result_data, result_len_max)
4      && \unrelated(result_data, &sams_other) && \unrelated(startpoints_data, &sams_other)
5      && 4 < l && (4 * l + 5) * startpoints_len <= result_len_max
6      && \valid(bufferradius)
7
8    @modifies sams_other, result_data[:(4 * l + 5) * startpoints_len],
9              *result_len, *bufferradius
10
11   @ensures \result == sams_safe -->
12     *result_len == (4 * l + 5) * startpoints_len &&
13     ${ let S = ^Vector2DList{startpoints_data, startpoints_len};
14            R = ^Vector2DList{result_data, *result_len};
15            ROBOT = conv S;
16            SAFETYZONE = extend_by_radius `{*radius} (conv R)
17        in  ∀ p ∈ ROBOT . ∀ s ∈ {`s_min .. `s_max} . ∀ a ∈ {`alpha_min .. `alpha_max}.
18            ( arc s a p ) ⊂ SAFETYZONE
19      }
20   @*/
21 SAMSStatus safetyzone_salpha_interval( Float32 s_min, Float32 s_max,
22                Float32 alpha_min, Float32 alpha_max, Int32 l,
23                const Vector2D * startpoints_data, Int32 startpoints_len,
24                Vector2D * result_data, Int32 result_len_max, Int32 * result_len, Float32 *radius );
```

Fig. 9 Actual specification source (with Isabelle pretty printing) of the function implementing step 2*, i.e. the extension of Fig. 8 to intervals in $(s,\alpha)^T$. Similar to Fig. 8 line 17 and 18 state that the arc which any point of the robot traverses stays inside the safety zone. But unlike Fig. 8 this is ensured for all s and α inside the input intervals. The other difference is the use of the buffer-radius in the definition of the safety zone in line 16 which simply was not needed in Fig. 8. Again, the use of high-order logic, i.e. quantification over infinite domains, allows a very concise and elegant specification.

4.4 The Verification Environment

To prove that a function satisfies a specification in the language sketched above, we implemented a verification environment based on Isabelle. Firstly, it contains a formalisation of the semantics of the subset of the C language used in our application (most of MISRA C) and the specification language in Isabelle. Secondly, its front-end reads and parses annotated C code and outputs Isabelle terms representing their abstract syntax. Finally, a set of automated proof procedures (called tactics) reduces annotated specifications. In short, when proving that a function satisfies a given postcondition and modification frame, we calculate the effect of the function's body on the postcondition by backwards reasoning in the Hoare style, and show that the transformed postcondition is implied by the precondition. Along the transformations, various conditions concerning array access and pointer dereferencing have to be proven. The set of tactics just mentioned does this automatically, leaving only the core of the correctness proof concerned with domain-level properties to the user.

Further details about the specification language and how functions can be proven correct in Isabelle w. r. t. their specification have been described in a previous paper (Lüth and Walter, 2009). The development of the verification environment was a substantial task, but it can now be reused for the verification of any MISRA C program, even in completely different domains.

The verification is modular: the correctness of each function is proven separately, assuming that all other functions satisfy their respective specifications. Not only is this crucial to keep the size of proofs at a manageable level, it moreover allows us to focus formal verification on those functions which are crucial to functional correctness; other functions may contain constructs that our tools cannot reason about, or may not pertain to global correctness (e. g., logging), and can be treated more adequately by manual review or functional tests.

4.5 Limitations of our Tool

Our tool focuses on functional correctness, and does not consider aspects such as execution time analysis and bounds, resource consumption, concurrency, and the interface between hardware and software. This is a clear separation of concerns, as it is becoming common consensus that only the use of multiple, specialised tools and methodologies can achieve a high level of confidence in software (Hoare, 2009). Moreover, we had to make some abstractions when modelling the C language: firstly, we do not model the whole language but rather the subset described by the MISRA programming guidelines (MISRA-C), and secondly, we abstract the respective machine data types to $\mathbb{Z}$ and $\mathbb{R}$ ignoring overflows and rounding errors (as common in formal verification). This is the price we had to pay to obtain a formalisation in which interesting, abstract, functional properties can be proved with tolerable effort.

4.6 Dynamic Analysis and Tests

Due to the use of formal verification, functional testing was not necessary which reduced the amount of tests that had to be performed to a significant extent. To ensure that (a) the occurring rounding errors do not violate specified error bounds and (b) no arithmetic under-/overflows occur, we performed testing.

In order to test for rounding errors, we analysed each function performing floating point operations numerically, and specified quantity dependent error bounds. From these, we derived test cases covering typical scenarios (e. g. driving straight forward) as well as boundary cases (e. g. driving maximum speed). For the actual tests, we gave a reference implementation, which was as an exact copy of the tested function carrying out each computation step with double precision, and compared the results of functions under test with those of the reference implementation. In order not to neglect potential error propagation and accumulation, integration testing was performed. We tested random cases and systematic border cases and determined the maximum error encountered. This procedure is established industrial practice. Of course, the numerical bound is not as reliable as the Isabelle proof of functional correctness in infinite precision computation. However, actually proving hard bounds on the numerical error in finite precision computation is extremely difficult and was beyond our scope.

At the end of the day, the numerical error bound is a choice with the specification of the system. Based on our analysis and practical considerations we specified the numerical error to be bound by 0.1% of the maximal coordinate in the safety zone. This was sufficient to make our test cases pass while yielding to an insignificant error in practice at the same time ($1mm$ error per $1m$ expansion of the safety zone). Nevertheless, we add this numerical error bound to the safety zone's buffer radius. This makes the overall algorithm conservative again, of course with the restriction discussed above.

Under- and overflow tests were performed at the unit level with sub-functions being replaced by appropriate stubs. All essential test cases were derived from those combinations of module-inputs and stub-outputs leading to smallest or largest (intermediate) computation results thereby allowing detection of potential under-/overflows in functions under test.

All tests were specified and executed with the test tool RT-Tester (2006) which provided very helpful features including automatic test documentation and report generation.

As opposed to the functional correctness verification, the numerical testing is accepted industry practice, showing that our method of functional verification can be seamlessly integrated with other verification methods covering different aspects of the verification process.

4.7 Related Work

There are two main approaches to software verification. Our language lies in the tradition of design by contract languages (Meyer, 1991), where programs are annotated with specifications. Other such languages include ACSL (Baudin et al, 2008) or VCC (Cohen et al, 2009) for C, JML (Burdy et al, 2005) for Java, or SPARK (Barnes, 2003) for Ada. However, our language additionally allows to include higher-order logic expressions in the syntax of the theorem prover Isabelle in specifications. This gain in language expressivity is the crucial ingredient for allowing more abstract specifications in which program values are put in relation to their corresponding domain values, e. g. specifications can involve infinite sets and other non-computable entities.

The other approach is to specify the intended behaviour of the system in a formalism different from the programming language. For controller software, differential equations can be used (e. g. MATLAB-Simulink). Another common formalism are state machines, as in StateCharts (Harel, 1987), UML State Diagrams, or Abstract State Machines (Gurevich, 2000). This approach can be extended to include temporal aspects (Bengtsson et al, 1995) or distributed systems as with CSP (Roscoe, 1998) or Spin (Holzmann, 2003). Hybrid systems as introduced by Henzinger (1997) combine discrete states and continuous variables described by differential equations. In that vein, Braman et al (2007) have used hybrid systems to verify the safety of goal-based robotic controllers, concretely a control system for an aerial mission on Saturn's moon Titan (Braman, 2009). Applications of hybrid systems to autonomous vehicles and cars include the early work by Dolginova and Lynch (1997) and Wongpiromsarn et al (2009), which is pen-and-paper based, and by Loos et al (2011), who use the KeYmaera system to formalise proofs. This system was also used by Platzer and Clarke (2009) to verify a flight collision avoidance system. The SCADE system uses the Lustre language, which combines state machines and synchronous data flow, and generates code from those specifications (Dion and Gartner, 2005). It is certified according to IEC 61508 up to SIL 3, and used mainly in aerospace. Orccad uses the related Esterel language to specify the coordination of real-time robotic controllers (Simon et al, 2006). All these approaches focus on the operational behaviour of the system. In robotics, the data are equally important, because of the rich domain (as mentioned above). Using a theorem prover to model the domain allows us to formalise it close to its textbook definitions; we do not have to translate the mathematics into another, less rich formalism, but instead can directly relate the domain model to the program code.

Bensalem et al (2009, 2010b) propose an interesting two-tier architecture for robotics applications, where components implemented with the model-based Genom tool are composed in the BIP framework, with safety properties such as absence of deadlocks proven by specialised tools such as D-Finder Bensalem et al (2010a). This is complementary to our work; the algorithm described here can very well form the core of a component in a framework like BIP.

Other examples of formally verified algorithms tend to be mathematically elegant but rather idealised to highlight the underlying scientific methods. For instance, Meikle and Fleuriot (2009) proved correctness of Graham scan, a convex hull algorithm which is actually one part of the final step in our algorithm. Compared to that, our collision avoidance algorithm is more complex and less clean, reflecting its real-world origins and applicability.

5 The Safety Zone Algorithm

This chapter provides the safety zone algorithm in detail. After presenting the algorithm in pseudo-code in Sec. 5.1, separate parts are introduced in detail following step-by-step the development stages and algorithm steps in Fig. 2: Sec. 5.2 covers the straight braking model (step 1b) and Sec. 5.3 the curved parts of the braking model (steps 1a and 1c). The safety zone computation (step 2) is valid for straight and curved motion and thus also introduced in Sec. 5.3. Then, Sec. 5.4 extends both operations to intervals (steps 1* and 2*). Finally, Sec. 5.5 presents the postprocessing extension (step 3*), which transforms the safety zone into a laser-scan representation.

5.1 Overview

Conventionally, the vehicle moves in a plane and therefore has the pose $(x, y, \alpha)^T$ and extended velocity $(v, \omega)^T$. $(x, y)^T$ is the position of the robot's reference point in world coordinates and α is the robot's orientation. v and ω are its translational and rotational velocities. The shape of the robot is given by a polygon, i. e. a list of n points $R_1, \ldots, R_n$ in the robot's egocentric coordinate system $\mathfrak{C}$. The robot's area $\mathfrak{R}$ is the convex hull of these points

$$\mathfrak{R} := \text{conv}\,\{R_i | i = 1, \ldots, n\} \tag{2}$$

The safety zone is computed in the robot's egocentric coordinate system at the time of braking start, which we call $\mathfrak{C}_0$.

The overall algorithm is given in Algo. 1. Our inter-

Algorithm 1 Safety zone computation (steps 1*-2*)

Configuration:
array of straight braking measurements $(v_1,s_1),\ldots,(v_m,s_m)$,
latency $\triangle t$,
array of robot contour points $R_1,\ldots,R_n$,
number of approximation points per circular arc L
Input:
bounds for translational velocity $v_{\min},v_{\max}$,
bounds for rotational velocity $\omega_{\min},\omega_{\max}$
Output:
array of points $W_1,\ldots,W_k$,
buffer radius q

1 *// candidates from eqs. (43)-(44)*
2 **for all** candidates (v^c,ω^c) of limiting configurations **do**
3 set vs $\leftarrow \sqrt{v^{c2}+D^2\omega^{c2}}$
4 *// compute characteristic time without dividing by vs=0*
5 **if** vs $\le v_1$ **then**
6 set ts $\leftarrow \frac{s_1}{v_1}$
7 **else**
8 find j such that $v_{j-1} <$ vs $\le v_j$
9 set ts $\leftarrow \frac{s_{j-1}+\frac{s_j-s_{j-1}}{v_j-v_{j-1}}(\text{vs}-v_{j-1})}{\text{vs}}$
10 set $s^c \leftarrow (\text{ts}+\triangle t)\,v^c$
11 set $\alpha^c \leftarrow (\text{ts}+\triangle t)\,\omega^c$
12 find $s_{\min} \leftarrow \min_c s^c$, $s_{\max} \leftarrow \max_c s^c$
13 find $\alpha_{\min} \leftarrow \min_c \alpha^c$, $\alpha_{\max} \leftarrow \max_c \alpha^c$

14 **for all** $i = 1$ **to** n **do**
15 set $W_i \leftarrow R_i$
16 set $k \leftarrow n$
17 *// four limiting movements*
18 **for all** $(s,\alpha) \in \{s_{\min},s_{\max}\}\times\{\alpha_{\min},\alpha_{\max}\}$ **do**
19 **for all** $i = 1$ **to** n **do**
20 *// $T(s,\alpha)$ from eq. (28)*
21 set V0 $\leftarrow R_i + \begin{pmatrix} 1 & \tan\alpha/2L \\ -\tan\alpha/2L & 1 \end{pmatrix} \cdot \frac{1}{2}(T(\frac{s}{L},\frac{\alpha}{L})\cdot R_i - R_i)$
22 **for all** $j = 0$ **to** $L-1$ **do**
23 set $W_{k+j+1} \leftarrow T(\frac{j\cdot s}{L},\frac{j\cdot\alpha}{L})\cdot$V0
24 set $W_{k+L+1} \leftarrow T(s,\alpha)R_i$
25 set $k \leftarrow k+L+1$
26 set $q \leftarrow \frac{1}{6}\left(\frac{\alpha_{\max}-\alpha_{\min}}{2}\right)^2 \max\{|s_{\max}|;|s_{\min}|\} + \left(1-\cos\frac{\alpha_{\max}-\alpha_{\min}}{2}\right)\max_{1\le i\le n}\{|R_i|\}$

27 **return** W_1,...,W_k, and q

nal representation of areas are Sphere Swept Convex Hulls (SSCH) as depicted in Fig. 3. They are internally stored as an unsorted array of points $[P_k]_{k=1}^K$ and a buffer radius q. Formally, the represented area is written as the Minkowski sum of the convex hull and a disk of radius q:

$$A\left([P_k]_{k=1}^K;q\right) = \left\{P+Q \;\middle|\; P\in\text{conv}\left\{[P_k]_{k=1}^K\right\}, |Q|\le q\right\}. \quad (3)$$

Such a representation facilitates safety related algorithms, because it doesn't have any connectivity information (vertices, edges) which can suffer from numerical problems such as (nearly) duplicate points.

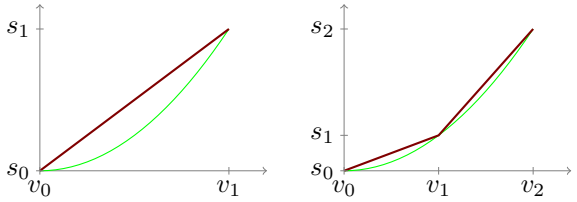


Fig. 10 Piecewise linear interpolation of the straight braking distance for $0 \le v \le v_m$: exemplary real braking distance (lower green line) and upper bound (upper red line) for one (left) resp. two breaking measurements (right).

5.2 Straight Motion

5.2.1 Step 1b - Straight Braking Model

The straight braking model (step 1b) computes an upper bound of the braking distance $s(v)$ for straight movements with velocity v. It is configured by a list $(v_1,s_1),\ldots,(v_m,s_m)$ of braking measurements. Each of the $m \ge 1$ measurements consists of a velocity v_j and the corresponding measured braking distance s_j. One measurement must be taken at maximum speed. That one measurement already establishes a complete configuration of the straight braking model but more measurements at lower speed are possible, making the model less conservative. So, the user can decide on the number of measurements depending of the effort of measuring straight braking distances for his vehicle. We assume the list of measurements to be sorted in ascending order and extended by $(v_0,s_0) = (0,0)$.

The straight braking model applies a piecewise linear interpolation (Fig. 10) and bounds the braking distance for velocity v with $0 \le v \le v_m$ by its neighbouring measurements (v_{j-1},s_{j-1}) and (v_j,s_j). It returns

$$\hat{s}(v) = s_{j-1} + \frac{s_j - s_{j-1}}{v_j - v_{j-1}}(v - v_{j-1}), \quad (4)$$

where j is determined to satisfy $v_{j-1} \le v \le v_j$. For velocities $v \ge v_m$ the braking distance is conservatively extrapolated as

$$\hat{s}(v) = \frac{s_m}{v_m^3}v^3. \quad (5)$$

The upper bound (4) can be derived from assumption (A3b). We examine the rate of kinetic energy dissipation

$$\dot{E} = m\,a\,v, \quad (6)$$

with m being the vehicle's mass, a its braking acceleration at time t, and $v \ge 0$ the velocity at the same time. We obtain

$$\frac{|\dot{E}|}{v^2} = m\frac{|a|}{v} \quad (7)$$

which only depends on the velocity ($a = a(v)$) because of (A3d). Thus, (A3b) is equivalent with $v/|a|$ being a monotonically increasing function. This yields a physical interpretation of restrictions given by (A3b) concerning the vehicle (cf. Fig. 4): The assumption holds for Coulomb friction

($a = \text{const}$) as encountered with usual brakes, proportional friction, for instance when short-circuiting a motor ($a \propto v$) but not for viscous friction ($a \propto v^2$), e. g. parachutes, airplanes or ships. So the assumption is valid for vehicles.

Further, because $s(t)$ is the remaining braking distance from t to standstill, we have

$$\frac{ds}{dv} = \frac{ds}{dt}\frac{dt}{dv} = \frac{-v}{a}. \tag{8}$$

By (7) and $a < 0$ (8) is monotonically increasing and therefore $s(v)$ is a convex function. This proves the upper bound (4) to be correct.

The upper bound (5) is a consequence of (A3c). It allows to extrapolate beyond the maximum measurement v_m. We need it because (4) is not valid for $v > v_m$ but the computations in Sec. 5.3 exceed v_m from time to time slightly. We omit the proof of (5) for brevity; it is very similar to the proof of (4).

5.3 Curved Motion

The curved braking model does not use any additional configuration parameters. It just requires the straight braking model to be configured, which is much simpler than obtaining curved braking measurements. Nevertheless, it can conservatively approximate braking trajectories of both types, straight lines and circular arcs.

5.3.1 Step 1a - Equivalent Straight Velocity

The equivalent straight velocity v_S of a curved motion input $(v,\omega)^T$ is the one having the same kinetic energy

$$E = \frac{1}{2}mv^2 + \frac{1}{2}J\omega^2 \tag{9}$$

as $(v,\omega)^T$ with m being mass and J being inertia. Because computing v_S would require the vehicle parameters m and J, we compute an upper bound

$$\hat{v}_S = \sqrt{v^2 + D^2\omega^2} \qquad \text{with } D = \max_i |R_i| \tag{10}$$

instead. The proof of (10) starts by representing m and J in terms of density ρ

$$m = \int \rho(x)\,dx \qquad J = \int \rho(x)\,x^2\,dx. \tag{11}$$

We obtain

$$D^2 m \geq J \tag{12}$$

because $D \geq |x|$ limits the robot dimension ($\rho(x) = 0$ for all $|x| > D$). Thus the computed straight velocity $\hat{v}_S$ has kinetic energy

$$E_S = \frac{1}{2}m\left(v^2 + D^2\omega^2\right) \geq E \tag{13}$$

and yields at least the characteristic time and braking distance of v_S.

5.3.2 Step 1c - Computing Braking Configuration $(s,\alpha)^T$

The final step of the curved braking model is computing the braking configuration $(s,\alpha)^T$ from the equivalent straight velocity v_S (result of 1a) and its corresponding braking distance $s(v_S)$ (result of 1b).

From (A2) we know that the vehicle has a fixed steering angle during the whole braking, so the curvature, i. e. the ratio between rotational and translational velocities, is constant over time

$$\frac{\omega(t)}{v(t)} = \frac{\omega}{v}. \tag{14}$$

As a consequence, both velocities must have the same characteristic function $\lambda(t)$ telling by which fraction they decay over time

$$\lambda(t) = \frac{v(t)}{v} = \frac{\omega(t)}{\omega}. \tag{15}$$

Now we consider the change over time of the vehicle's kinetic energy during brakings from $(v,\omega)^T$ resp. v_S

$$E(t) = \frac{1}{2}m\,v(t)^2 + \frac{1}{2}J\omega(t)^2 = \lambda(t)^2 E \tag{16}$$

$$E_S(t) = \frac{1}{2}m\,v_S(t)^2 = \lambda_S(t)^2 E_S \tag{17}$$

with λ_S being the characteristic function of the equivalent straight braking. As the kinetic energies of $(v,\omega)^T$ and v_S are equal at braking start, their change over time is according to (A3a) also equal, thus

$$E_S(t) = E(t) \tag{18}$$

and consequently

$$\lambda_S(t) = \lambda(t). \tag{19}$$

With v_S and $s_S = s(v_S)$ being the startup velocity and braking distance of the straight braking, integration over t shows

$$\frac{s_S}{v_S} = \frac{s}{v} = \frac{\alpha}{\omega} = \int_0^T \lambda(t)dt. \tag{20}$$

From the convexity of the function $s(v)$ (cf. Sec. 5.3.1) we conclude that for all upper bounds $\hat{v}_S \geq v_S$ the following inequality holds

$$\frac{\hat{s}(\hat{v}_S)}{\hat{v}_S} \geq \frac{s(v_S)}{v_S} \tag{21}$$

and we get

$$v\frac{\hat{s}(\hat{v}_S)}{\hat{v}_S} \geq s \qquad \omega\frac{\hat{s}(\hat{v}_S)}{\hat{v}_S} \geq \alpha \tag{22}$$

which justifies the pure braking distance without latencies

$$\begin{pmatrix}\hat{s}\\ \hat{\alpha}\end{pmatrix} = \frac{\hat{s}(\hat{v}_S)}{\hat{v}_S}\begin{pmatrix}v\\ \omega\end{pmatrix}. \tag{23}$$

The full braking model computation extends the result of (23) by the time delay $\triangle t$ (latency) of driving with constant velocity $(v,\omega)^T$ before the braking really starts

$$\mathrm{BM}_{\triangle t}(v,\omega) = \begin{pmatrix} \hat{s} \\ \hat{\alpha} \end{pmatrix} = \begin{pmatrix} \frac{\hat{s}(\hat{v}_S)}{\hat{v}_S} + \triangle t \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix}. \tag{24}$$

An important property is that the curvature of the resulting approximation $(\hat{s},\hat{\alpha})^T$ is equal to the curvature given by the configuration $(v,\omega)^T$

$$\frac{\hat{\alpha}}{\hat{s}} = \frac{\omega}{v}. \tag{25}$$

This is satisfied by (23) as well as (24).

5.3.3 Step 2 - Safety Zone Computation

The outcome of the braking model are arc length s and corresponding angle α (Fig. 11a). This $(s,\alpha)^T$ representation jointly models circular trajectories ($s \neq 0$, $\alpha \neq 0$) as well as straight trajectories ($s \neq 0$, $\alpha = 0$) and even turning on the spot ($s = 0$, $\alpha \neq 0$). Using s and α all necessary kinds of trajectories and transitions between them are modeled without singularities. For any point $\lambda \in [0,1]$ on the trajectory the robot pose is given by orientation $\alpha_\lambda = \lambda\alpha$ and reference point position $(x_\lambda, y_\lambda)^T$ which we obtain from the arc length up to that point $s_\lambda = \lambda s$ by

$$\begin{pmatrix} x_\lambda \\ y_\lambda \end{pmatrix} = s_\lambda \operatorname{sinc}\frac{\alpha_\lambda}{2} \begin{pmatrix} \cos\frac{\alpha_\lambda}{2} \\ \sin\frac{\alpha_\lambda}{2} \end{pmatrix} \tag{26}$$

with $\operatorname{sinc}\phi$ being the sinus cardinalis

$$\operatorname{sinc}\phi = \begin{cases} \frac{\sin\phi}{\phi} & \phi \neq 0 \\ 1 & \phi = 0 \end{cases}. \tag{27}$$

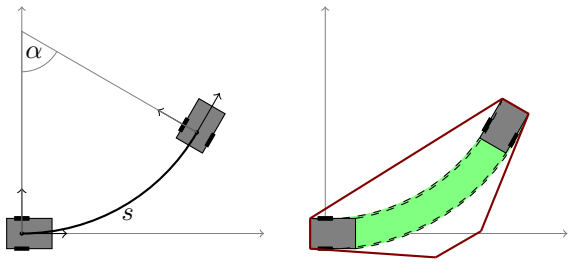
Coordinates of (26) are in the egocentric coordinate system at braking start $\mathfrak{C}_0$, which also is the base coordinate system of the whole safety zone computation, thus of all point vectors in the following that are not explicitly bound differently. The egocentric coordinate system at λ, denoted $\mathfrak{C}_\lambda$, is given by the robot pose. Hence,

$$T(s_\lambda,\alpha_\lambda) = \begin{pmatrix} \cos\alpha_\lambda & -\sin\alpha_\lambda & s_\lambda \operatorname{sinc}\frac{\alpha_\lambda}{2}\cos\frac{\alpha_\lambda}{2} \\ \sin\alpha_\lambda & \cos\alpha_\lambda & s_\lambda \operatorname{sinc}\frac{\alpha_\lambda}{2}\sin\frac{\alpha_\lambda}{2} \end{pmatrix} \tag{28}$$

yields a transformation of robot points from $\mathfrak{C}_\lambda$ into $\mathfrak{C}_0$. We assume that the input point is extended by the component 1 to be in homogeneous coordinates always before applying the transformation T. For readability we omit that in our formulae.

Using (28) the requested braking area $\mathfrak{H}$ (Fig. 11b) is defined as the union of the vehicle shape as it moves along the braking trajectory

$$\mathfrak{H}(s,\alpha) := \bigcup_{\lambda\in[0,1]} \{T(\lambda s,\lambda\alpha)\cdot R \mid R \in \mathfrak{R}\}. \tag{29}$$



(a) (b)

Fig. 11 (a) Braking configuration $(s,\alpha)^T$ is the input of the safety zone computation (step 2). (b) Braking area (green area), trajectories of the four robot contour vertices (dashed arcs), and safety zone (red polygon) for the braking configuration from (a). Note that the actual implementation uses more points for a tighter approximation.

The rigid body transformation T is linear and commutes with taking the convex hull of $\mathfrak{R} = \operatorname{conv}\{[R_i]_{i=1}^n\}$. Thus,

$$\mathfrak{H}(s,\alpha) = \bigcup_{\lambda\in[0,1]} \operatorname{conv}\{T(\lambda s,\lambda\alpha)\cdot R_i \mid i = 1,\ldots,n\}. \tag{30}$$

The safety zone computation (step 2) determines a superset

$$H(s,\alpha) \supseteq \operatorname{conv}\mathfrak{H}(s,\alpha), \tag{31}$$

of the convex hull of that braking area. Actually, a superset of the braking area would be sufficient, but the convex hull is intrinsic to our internal area representation.

The computation of $H(s,\alpha)$ exploits a general property of convex hulls

$$\operatorname{conv} S_1 \cup \operatorname{conv} S_2 \subseteq \operatorname{conv}(S_1 \cup S_2), \tag{32}$$

the union of convex hulls is contained in the convex hull of the union. From (30) that yields

$$\mathfrak{H}(s,\alpha) \subseteq \operatorname{conv} \bigcup_{\lambda\in[0,1]} \{T(\lambda s,\lambda\alpha)\cdot R_i \mid i = 1,\ldots,n\} \tag{33}$$

$$= \operatorname{conv} \bigcup_{i=1}^{n} \{T(\lambda s,\lambda\alpha)\cdot R_i \mid \lambda \in [0,1]\} \tag{34}$$

and establishes the basic principle of our safety zone algorithm: *First compute the circular arcs for each R_i (or a superset) and afterwards take the convex hull of all of them* (cf. Fig. 11b).

The trajectory of each robot contour point R_i is approximated separately. By (28) all of these trajectories are circular arcs. Each of them is covered by a area containing the entire arc. Four possible ways of doing that are shown in Fig. 12. The circular arc $\{T(s_\lambda,\alpha_\lambda)R_i|\lambda\in[0,1]\}$ starts at $P_i^1 = R_i$ and stops at $P_i^2 = T(s,\alpha)R_i$. All of the areas given in Fig. 12 are supersets of that trajectory.

In our implementation we use a generalised version of Fig. 12d that separates the circular arc into L equal parts and covers each of them by three points as shown in Fig. 12c. The full arc is of course covered by the convex hull of all

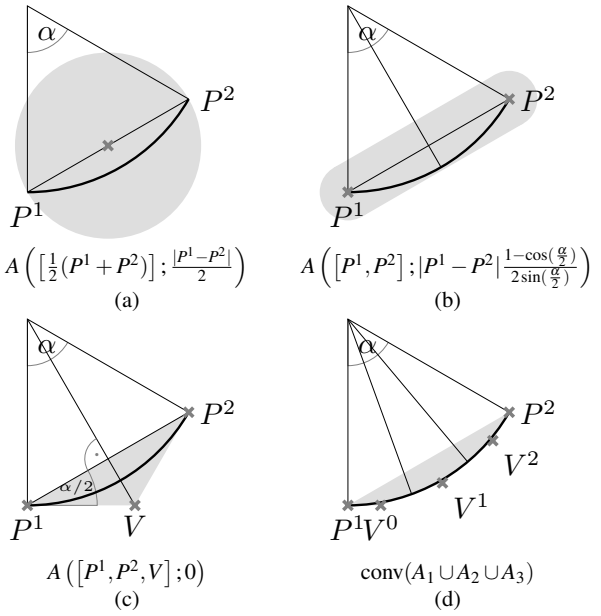


Fig. 12 Approximation of the convex hull of a circular arc using one (a), two (b), or three (c) generating points and an appropriate radius. $V = P^1 + Q(\alpha)\frac{1}{2}(P^2 - P^1)$ is the intersection point of the tangents in P^1 and P^2. In (d) the arc is split into three equal parts and then (c) is applied to each.

these points. But the intermediate start and end points of the subarcs are actually not needed as they lie in the convex hull of the start and end point together with the tangent intersections V_i^j of the subarcs (Fig. 12d). That yields an approximation using $L+2$ points per circular arc:

$$\{T(s_\lambda, \alpha_\lambda)R_i | \lambda \in [0,1]\} \subseteq A\left(\left[P_i^1, P_i^2, [V_i^j]_{j=0}^{L-1}\right]; 0\right) \tag{35}$$

with

$$V_i^j = T(\tfrac{j \cdot s}{L}, \tfrac{j \cdot \alpha}{L}) \cdot U_i^3 \text{ and} \tag{36}$$

$$U_i^1 = R_i \tag{37}$$

$$U_i^2 = T(\tfrac{s}{L}, \tfrac{\alpha}{L}) \cdot R_i \tag{38}$$

$$U_i^3 = U_i^1 + Q(\tfrac{\alpha}{L})\tfrac{1}{2}(U_i^2 - U_i^1) \tag{39}$$

$$Q(\alpha) = \begin{pmatrix} 1 & \tan\frac{\alpha}{2} \\ -\tan\frac{\alpha}{2} & 1 \end{pmatrix} \tag{40}$$

where U_i^1, U_i^2, U_i^3 are the approximation points of the first subarc. U_i^1 and U_i^2 are start and endpoint of this subarc and U_i^3 the intersection point of the tangents in U_i^1 and U_i^2 which can be computed by Q. Finally, V_i^j are the transformations of the tangent intersection point U_i^3 into all other subarcs. Notice that (35)-(40) as well as the areas in Fig. 12 also work properly in the case of a straight trajectory. For brevity we omit the proof of (35) here, it is based on the idea that both tangents in Fig. 12c as well as the chord are bounds of the circular arc.

Finally, the safety zone for $(s,\alpha)^T$ arises when the circular arc approximations for all vertices R_i of the robot are united and their convex hull is taken

$$H(s,\alpha) = A\left(\left[P_i^1, P_i^2, [V_i^j]_{j=0}^{L-1}\right]_{i=0}^{n}; 0\right). \tag{41}$$

5.4 Extension to Input Intervals

In practice, measurement errors (in v and ω) are unavoidable, thus a safety zone H computed for just a single configuration $(v,\omega)^T$ as shown in Sec. 5.3 will never be correct. For this reason, we extended the curved braking model (steps 1a-c) and the safety zone computation (step 2) to interval operations (step 1* and 2*) allowing a user to add potential measurement and modelling errors and call the algorithm with inputs that safely cover the true configuration.

5.4.1 Step 1 - Curved Braking Model of Input Intervals*

Step 1* is the extension of steps 1a-c to intervals. It takes an interval $[v_{\min}, v_{\max}] \times [\omega_{\min}, \omega_{\max}]$ instead of a single $(v,\omega)^T$ and maps it to an interval $[s_{\min}, s_{\max}] \times [\alpha_{\min}, \alpha_{\max}]$. As the mapping is non-linear the computed interval is not exact but conservatively bounds the complex shaped exact result in (s,α) space (cf. Fig. 2, lower row, middle diagram).

The bound is obtained from candidates $(v^c, \omega^c)^T$ of limiting configurations

$$(v^c, \omega^c)^T \in V \times \Omega \tag{42}$$

with

$$V = \begin{cases} \{v_{\min}, 0, v_{\max}\} & 0 \in [v_{\min}, v_{\max}] \\ \{v_{\min}, v_{\max}\} & 0 \notin [v_{\min}, v_{\max}] \end{cases} \tag{43}$$

$$\Omega = \begin{cases} \{\omega_{\min}, 0, \omega_{\max}\} & 0 \in [\omega_{\min}, \omega_{\max}] \\ \{\omega_{\min}, \omega_{\max}\} & 0 \notin [\omega_{\min}, \omega_{\max}] \end{cases} \tag{44}$$

by applying step 1a-c to every one of them

$$(s^c, \alpha^c)^T = \mathrm{BM}_{\triangle t}(v^c, \omega^c) \tag{45}$$

and finding the minimum and maximum values

$$s_{\min} = \min_c s^c \qquad s_{\max} = \max_c s^c \tag{46}$$

$$\alpha_{\min} = \min_c \alpha^c \qquad \alpha_{\max} = \max_c \alpha^c. \tag{47}$$

The proof of step 1* essentially shows that for every $(v,\omega)^T$ from $[v_{\min}, v_{\max}] \times [\omega_{\min}, \omega_{\max}]$ its braking model result $(s,\alpha)^T = \mathrm{BM}_{\triangle t}(v,\omega)$ lies within $[s_{\min}, s_{\max}] \times [\alpha_{\min}, \alpha_{\max}]$. (43) and (44) are based on the fact that (10) and (24) are monotonic for fixed signs of v and ω (in each quadrant of (v,ω)-space). We omit the detailed proof as it is just an exercise in analysis.

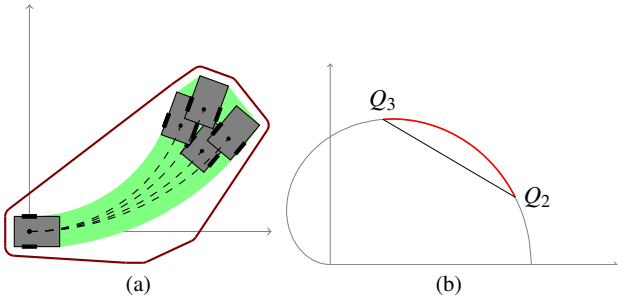


Fig. 13 (a) The four limiting movements (dashed lines), braking area (green area), and safety zone (outer red line) for the braking configurations interval $[s_{\min}, s_{\max}] \times [\alpha_{\min}, \alpha_{\max}]$. (b) Plot of $\mathbf{X}^A_{\lambda,R,s_\mu}(\alpha_\gamma)$ with Q_2 and Q_3 at α_0 resp. α_1: the function is not linear, the distance to the line segment Q_2Q_3 is bounded by q^A in (63).

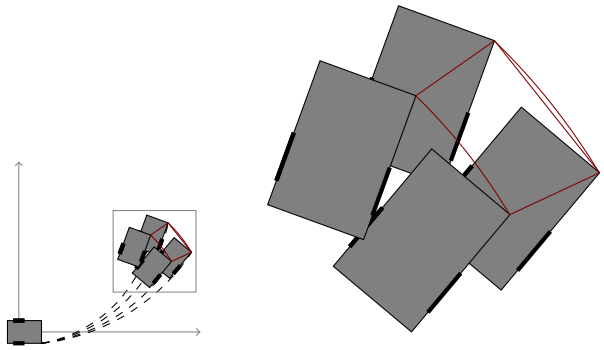


Fig. 14 The location $\mathbf{X}_{1,R_i}(s_\mu, \alpha_\gamma)$ along the borders $\mu = 0$, $\gamma = 0$, $\mu = 1$, and $\gamma = 1$ of $[s_{\min}, s_{\max}] \times [\alpha_{\min}, \alpha_{\max}]$ with R_i being the lower right vertex of the robot contour (close red curve). The location $\mathbf{X}^*_{1,R_i}(s_1, \alpha_\gamma)$ is shown for $\gamma \in [0,1]$ (red line segment).

5.4.2 Step 2* - Safety Zone of Input Intervals

Step 2* extends step 2 to an input interval $I = [s_{\min}, s_{\max}] \times [\alpha_{\min}, \alpha_{\max}]$. Its result $H(s_{\min}, s_{\max}, \alpha_{\min}, \alpha_{\max})$ is a superset of the convex hull of the braking area for all braking configurations $(s,\alpha)^T \in I$

$$H(s_{\min}, s_{\max}, \alpha_{\min}, \alpha_{\max}) \supseteq \bigcup_{(s,\alpha)^T \in I} \operatorname{conv} \mathfrak{H}(s,\alpha) \tag{48}$$

The computation works as follows: First, the safety zones for the four limiting movements, i.e. the extreme cases $H_1 = H(s_{\min}, \alpha_{\min})$, $H_2 = H(s_{\max}, \alpha_{\min})$, $H_3 = H(s_{\max}, \alpha_{\max})$, and $H_4 = H(s_{\min}, \alpha_{\max})$, are computed (Fig. 13a). And then, the convex hull of their union (later on called $\mathfrak{A}$) is expanded by an appropriate radius q bounding the non-linear effects of $T(s,\alpha)$. This results in the overall computation

$$H(s_{\min}, s_{\max}, \alpha_{\min}, \alpha_{\max}) = A\left(\left[\left[\left[P^1_{i,s,\alpha}, P^2_{i,s,\alpha}, [V^j_{i,s,\alpha}]_{j=0}^{L-1}\right]_{i=0}^{n}\right]_{s_{\max}}^{s_{\min},}\right]_{\alpha_{\max}}^{\alpha_{\min},}; q\right), \tag{49}$$

with

$$q = q^A + q^B \tag{50}$$

$$q^A = \tfrac{1}{6}\left(\tfrac{\alpha_{\max}-\alpha_{\min}}{2}\right)^2 \max\{|s_{\max}|; |s_{\min}|\} \tag{51}$$

$$q^B = \left(1 - \cos\tfrac{\alpha_{\max}-\alpha_{\min}}{2}\right) \max_{1\le i\le n}\{|R_i|\}. \tag{52}$$

The proof of (49)-(52) is more technical, because it is based on algebraic calculations rather than mathematical arguments as in the proofs so far. It analyzes the location of an arbitrary contour point $R \in \mathfrak{R}$ for an arbitrary braking configuration $(s_\mu, \alpha_\gamma) \in [s_{\min}, s_{\max}] \times [\alpha_{\min}, \alpha_{\max}]$ at an arbitrary time $\lambda \in [0,1]$ using (28)

$$\mathbf{X}_{\lambda,R}(s_\mu, \alpha_\gamma) = T(\lambda s_\mu, \lambda\alpha_\gamma)R \tag{53}$$

$$= \lambda s_\mu \operatorname{sinc}\tfrac{\lambda\alpha_\gamma}{2}\begin{pmatrix}\cos\frac{\lambda\alpha_\gamma}{2}\\ \sin\frac{\lambda\alpha_\gamma}{2}\end{pmatrix} + \begin{pmatrix}\cos\lambda\alpha_\gamma & -\sin\lambda\alpha_\gamma\\ \sin\lambda\alpha_\gamma & \cos\lambda\alpha_\gamma\end{pmatrix} R \tag{54}$$

with

$$s_\mu = (1-\mu)s_{\min} + \mu s_{\max} \qquad \mu \in [0,1] \tag{55}$$

$$\alpha_\gamma = (1-\gamma)\alpha_{\min} + \gamma\alpha_{\max} \qquad \gamma \in [0,1] \tag{56}$$

We call the convex hull of the union of the safety zones of the four extreme cases $\mathfrak{A} = \operatorname{conv}(H_1 \cup H_2 \cup H_3 \cup H_4)$. $\mathfrak{A}$ contains $\mathbf{X}_{\lambda,R}(s_\mu, \alpha_\gamma)$ for $\mu \in \{0,1\}$ and $\gamma \in \{0,1\}$. Thus,

$$\forall \mu \in [0,1] : \mathbf{X}_{\lambda,R}(s_\mu, \alpha_0) \in \mathfrak{A} \wedge \mathbf{X}_{\lambda,R}(s_\mu, \alpha_1) \in \mathfrak{A} \tag{57}$$

because $\mathbf{X}_{\lambda,R}(s_\mu, \alpha_\gamma)$ is a linear function in s. Now let

$$\mathbf{X}^*_{\lambda,R}(s_\mu, \alpha_\gamma) = (1-\gamma)\mathbf{X}_{\lambda,R}(s_\mu, \alpha_0) + \gamma\mathbf{X}_{\lambda,R}(s_\mu, \alpha_1) \tag{58}$$

be the linear connection between $\mathbf{X}_{\lambda,R}(s_\mu, \alpha_0)$ and $\mathbf{X}_{\lambda,R}(s_\mu, \alpha_1)$. Because $\mathbf{X}^*$ is linear it is contained in the convex $\mathfrak{A}$ (cf. Fig. 14).

$$\forall \mu \in [0,1]\ \forall \gamma \in [0,1] : \mathbf{X}^*_{\lambda,R}(s_\mu, \alpha_\gamma) \in \mathfrak{A} \tag{59}$$

The overall proof of $\mathbf{X}_{\lambda,R}(s_\mu, \alpha_\gamma) \in H(s_{\min}, s_{\max}, \alpha_{\min}, \alpha_{\max})$ now reduces to

$$\forall \mu \in [0,1]\ \forall \gamma \in [0,1] : \left\| \mathbf{X}_{\lambda,R}(s_\mu, \alpha_\gamma) - \mathbf{X}^*_{\lambda,R}(s_\mu, \alpha_\gamma) \right\| \le q, \tag{60}$$

i.e. to bounding the distance between $\mathbf{X}$ and $\mathbf{X}^*$ by q. The proof of (60) handles the two summands of (54) separately

$$\mathbf{X}^A_{\lambda,R,s_\mu}(\alpha_\gamma) = \lambda s_\mu \operatorname{sinc}\tfrac{\lambda\alpha_\gamma}{2}\begin{pmatrix}\cos\frac{\lambda\alpha_\gamma}{2}\\ \sin\frac{\lambda\alpha_\gamma}{2}\end{pmatrix} \tag{61}$$

$$\mathbf{X}^B_{\lambda,R,s_\mu}(\alpha_\gamma) = \begin{pmatrix}\cos\lambda\alpha_\gamma & -\sin\lambda\alpha_\gamma\\ \sin\lambda\alpha_\gamma & \cos\lambda\alpha_\gamma\end{pmatrix} R \tag{62}$$

and shows $\forall\lambda \in [0,1], \forall R \in \mathfrak{R}, \forall\mu \in [0,1]$

$$\forall\gamma \in [0,1]:$$
$$\left\| \mathbf{X}^A_{\lambda,R,s_\mu}(\alpha_\gamma) - \left((1-\gamma)\mathbf{X}^A_{\lambda,R,s_\mu}(\alpha_0) + \gamma\mathbf{X}^A_{\lambda,R,s_\mu}(\alpha_1)\right) \right\| \leq q^A \tag{63}$$
$$\left\| \mathbf{X}^B_{\lambda,R,s_\mu}(\alpha_\gamma) - \left((1-\gamma)\mathbf{X}^B_{\lambda,R,s_\mu}(\alpha_0) + \gamma\mathbf{X}^B_{\lambda,R,s_\mu}(\alpha_1)\right) \right\| \leq q^B \tag{64}$$

$\mathbf{X}^A_{\lambda,R,s_\mu}(\alpha_\gamma)$ and the bound (63) are visualized in Fig. 13b. Its proof applies a theorem, which itself is not trivial to prove: Let $\mathbf{Y}: [0;1] \to \mathbb{R}^2$ be two times differentiable and $\mathbf{Y}(0) = \mathbf{Y}(1) = (0,0)^T$, then

$$\forall\gamma \in [0;1]: \ \|\mathbf{Y}(\gamma)\| \leq \frac{1}{8} \max_{0\leq\tau\leq1} \left\|\mathbf{Y}''(\tau)\right\|. \tag{65}$$

This is the multidimensional extension of a well-known one-dimensional theorem. By applying that theorem and some lengthy calculations (63) reduces to

$$\left(\frac{\beta-\sin\beta}{\beta^2} - \frac{1-\cos\beta}{\beta}\right)^2 + \left(-\frac{\sin\beta}{\beta} + \frac{1-\cos\beta}{\beta^2} - \frac{\beta-\sin\beta}{\beta^3}\right)^2 \leq \frac{4}{9}, \tag{66}$$

which we proved by substituting the domain-level power-series based bounds (1) for sine and cosine.

Finally, $\mathbf{X}^B_{\lambda,R,s_\mu}$ in (62) is the change of the robot's orientation. So (64) is correct because q^B in (52) is a second order bound of a circular arc's distance to its corresponding chord.

The complicated proof sketched in this subsection was formally verified in Isabelle, which increases the confidence in (49) compared to a pen-and-paper proof.

5.5 Postprocessing Extension

Up to step 2*, we computed the safety zone in its internal representation. It is given by an unsorted list of points $[W_k]_{k=1}^K$ resulting from (49) and a buffer radius q

$$H(s_{\min}, s_{\max}, \alpha_{\min}, \alpha_{\max}) = A\left([W_k]_{k=1}^K; q\right). \tag{67}$$

To check the safety zone for obstacles, it is first transformed from $\mathfrak{C}_0$ into scanner coordinates $\mathfrak{C}_S$. This is done by applying $T_{S\leftarrow0}$ to each point W_k separately. Then, we apply the standard convex hull algorithm Graham-scan to the list of transformed points. The result is a counterclockwise sorted array of points $[P_i]_{i=1}^{n_G}$ representing the corners of a convex polygon:

$$[P_i]_{i=1}^{n_G} = \text{GRAHAM\_SCAN}\left([T_{S\leftarrow0}W_k]_{k=1}^K\right). \tag{68}$$

The safety zone (in $\mathfrak{C}_S$) is the expansion of that polygon by the buffer radius q, which we call buffered polygon. It is still given by $A\left([P_i]_{i=1}^{n_G}; q\right)$. Its boundary consists of edges $e^+_{i,i+1}$ and arcs b^+_i, which are the extensions of the edges $e_{i,i+1}$ and the vertices P_i of the polygon.

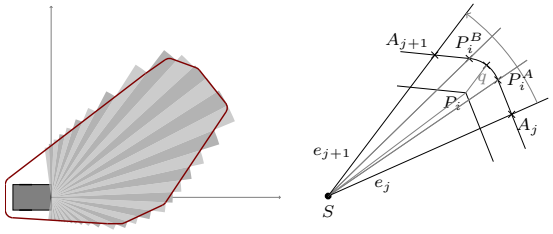


Fig. 15 Converting the buffered polygon into a laser scan representation. Coordinates $[P_i]_{i=1}^{n_G}$ are shifted such that the center of the laser ranger finder S is the origin.

Finally, we separate the buffered polygon $A\left([P_i]_{i=1}^{n_G}; q\right)$ into sectors using the fixed angular resolution of the laser rangefinder. In each sector we determine the maximum radial extend of the buffered polygon (Fig. 15). The result is an array of distances

$$[d_j]_{j=1}^{n_L} = \text{SAMPLING}\left([P_i]_{i=1}^{n_G}; q\right). \tag{69}$$

This is the laser scan representation of the computed safety zone limited to the field of vision of the laser rangefinder. We call this array the minimal laser scan. For each laser beam it contains the minimal distance that has to be free of obstacles, thus the minimal measurement into this direction that is safe. If any of the distances in the measured laser scan is less than its corresponding entry in the minimal laser scan the vehicle has to stop immediately.

The SAMPLING algorithm that transforms the buffered polygon into the minimal laser scan uses a sweep line principle. The sweep line is a ray starting at the origin of $\mathfrak{C}_S$ that rotates from the right to the left border of the laser rangefinder's field of view. Along the way the sweep line passes all sectors of the laser scan. At the same time it traverses edges and arcs of the buffered polygon and stores their maximum radial expansion within the current sector. That sector maximum is determined as the maximum of the distances of the buffered polygon at the left and right sector boundary and the maximum distance of all arcs of the buffered polygon that are completely contained in the sector. The distances at a sector boundary is the distance of its intersection point with either an edge or an arc. The distance of every point on the arc to the origin is bounded by $|P_i| + q$. Intersections with edges are computed explicitly except for numerical special cases, which are handled conservatively.

Correctness of the SAMPLING algorithm can be shown as follows: The triangle inequality shows that no point on the circular arc b^+_i can have a distance greater than $|P_i - S| + q$ with S being the origin within this computation (cf. Fig. 15). For every line segment the maximum distance is obtained at one of its endpoints, e. g. A_j or P^A_i. Thus, for each sector we only have to consider the distances at the sector boundaries,

endpoints of edges contained in the sector and all arc points inside the sector. As every endpoint of an edge also is a point of the connected arc its distance already is contained in the bound of the arc. Thus we can omit handling endpoints of edges and only have to consider intersections of edges at the sector boundaries. If the sector boundary intersects an arc, we handle the full arcs as described.

5.6 Summary

In this section, we have described our algorithm and its proof in the high-level language of textbook mathematics. Most of them have been implemented formally in Isabelle. To be explicit: Formal proofs of step 1 show the correct implementation of the braking model formulae, whereas a pen-and-paper proof shows the correctness of the formulae itself (parts of it also formalized in domain model). Step 2 was proven formally according to the specification in Fig. 8. For all the extensions proposed most of the separate functions were proven formally (74% percent), in particular (50) and everything below. There are two important functions that have been verified by testing instead of a formal proof: the implementation of step 3*, and as a consequence the overall main function containing all steps 1* - 3*. A function as displayed in Fig. 9 is not formulated explicitly in the code, we put it up to show the specification of step 2* without actually verifying this concrete specification. Not formally verifying the extension functions was an agreement with the certification authority for the purpose of saving time as the project drew to an end, all functions have been proven correctly on paper. The C sources containing implementation and specification and the verification tool are provided for download at `http://www.dfki.de/cps/sams/`.

6 Experiments

6.1 Demonstrator

In the SAMS project the certified algorithm, more precisely its certified C implementation, was intended as a generic component for safeguarding arbitrary AGVs. We have built a small demonstrator vehicle to experiment with our implementation in practice, which is shown in Fig. 1 and also in the supplemental video: The demonstrator drives through a cluttered environment, being stopped by our collision avoidance implementation whenever necessary. Simultaneously the video[5] shows the safety zones computed in real-time. Note that the demonstrator is an example for the system integration, but was not subject of the certification.

[5] The slightly jerky movements of the demonstrator are due to control issues, not emergency stops triggered by the safety component.

6.2 Simulator

While the demonstrator shows that the system works in practice, we rely on a physically realistic simulation for a more detailed analysis, as here ground-truth is available. Since it is just a tool for evaluation, we describe the simulator here only briefly for reference:

Like the algorithm, the simulator assumes that the vehicle retains its circular trajectory so this aspect is assumed and not verified. The braking behaviour however, is simulated physically correctly by considering torques and moment of inertia around the center of rotation (x_c, y_c), as defined by the current curvature $\frac{\omega}{v}$.

$$x_c = 0, \quad y_c = \frac{v}{\omega} \tag{70}$$

The moment of inertia around (x_c, y_c) is obtained from the robot shape $\mathfrak{R}$ and the robot mass m as

$$J = m \frac{\int_{(x,y)^T \in \mathfrak{R}} \left((x-x_c)^2 + (y-y_c)^2\right) dxdy}{\int_{(x,y)^T \in \mathfrak{R}} 1 dxdy}, \tag{71}$$

assuming constant mass density $\rho(x)$. Each brake i, located at (b_{ix}, b_{iy}) provides a Coulomb-friction force of f_i against the direction of motion $\operatorname{sgn} \dot{\alpha}$. Having a lever of $r_i = \sqrt{(b_{ix}-x_c)^2 + (b_{iy}-y_c)^2}$ the brake applies a torque of $-\operatorname{sgn} \dot{\alpha} f_i r_i$. So the overall vehicle dynamics is

$$\ddot{\alpha} = -\operatorname{sgn} \dot{\alpha} J^{-1} \sum_i f_i \sqrt{(b_{ix}-x_c)^2 + (b_{iy}-y_c)^2}, \tag{72}$$

$$s = \alpha \frac{v}{\omega} \tag{73}$$

With this simulation we can experimentally verify that the algorithm is conservative as proven in the software verification and by which factor the area is over-estimated. We also analyse how much the different sources of overestimation contribute to this factor. These sources are

- the linear interpolation (4) in the model for straight braking,
- the worst case bound (24) for rotational braking as a function of straight braking,
- the approximation (35) of an arc by a polygon, including the convex fill-in on the inner side of the arc,
- the use of an interval $[v_{\min}, v_{\max}] \times [\omega_{\min}, \omega_{\max}]$ instead of a single $(v, \omega)^T$ in (49) to reflect measurement uncertainty.

6.3 Braking Model

Fig. 17 shows how much the straight braking distance (4) is overestimated depending on how many measurements are used to configure the braking model. With a single measurement the factor is significant (39%), with two measurents

```
1  /*@
2    @requires directions_len - 1 <= result_len_max && 0 <= rbuffer &&
3              ${let V = ^Vector2DList{directions_data, directions_len}
4                 in (0 ∈ conv ^Vector2DSet{polygon_data, polygon_len}) ∧
5                    (sorted-by (λv1 v2. angle v1 v2 ∈ {φ. 0 < φ ∧ φ < π}) V) ∧ (∀v ∈ set V. |v| = 1) }
6             && \unrelated(polygon_data, polygon_len, result_data, result_len_max)
7             && \unrelated(directions_data, directions_len, result_data, result_len_max)
8    @modifies result_data[:directions_len-1]
9    @ensures \result == sams_safe -->
10     ${let E = ^RZList{result_data, directions_len - 1};
11           P = ^Vector2DList{polygon_data, polygon_len};
12           R = ^Vector2DList{directions_data, directions_len};
13           SAFETYZONE = extend-by-radius `radius (convex-area P)
14       in SAFETYZONE ∩ (angle-sector ^Vector2DR{&directions_data[0]}
15                        ^Vector2DR{&directions_data[directions_len-1]})
16            ⊆ scan-field R E }
17   @*/
18 SAMSStatus sampling( const Vector2D * polygon_data, Int32 polygon_len,
19                      Float32 radius,
20                      const Vector2D * directions_data, Int32 directions_len,
21                      Int32 * result_data, Int32 result_len_max );
```

Fig. 16 Actual specification source (with Isabelle pretty printing) of the function computing the laser scan representation. Fig. 16 shows the formal specification of the function that transforms a buffered polygon into a minimal laser scan. *directions-data* contains the *directions-length* sector boundaries whereas the result is of size $n_L = \textit{directions-length} - 1$ (line 8). As a precondition the sector boundaries must be given as unit length vectors in counterclockwise order (line 5). Further, the origin of the used coordinate system ($\mathfrak{C}_S$) has to be contained in the area of the polygon (line 4). If the function returns without error, it assures that the area corresponding to the minimal laser scan (line 16) is a superset of the area of the buffered polygon (line 13) limited to the field of vision of the laser rangefinder (lines 14–15).

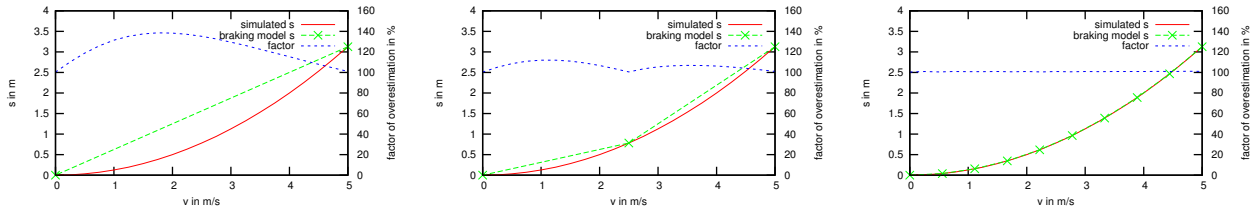


Fig. 17 Overestimation of the braking distance in straight motion. The plots show the true (simulated) braking distance s as a function of v and the linear interpolated result of our model (4) when configured with one (left), two (middle), and nine (right) straight braking measurements. The overestimation factor $\frac{A_{SAMS}}{A_{Sim}}$ shows how much larger the area of the computed safety zone is compared to the true braking area.

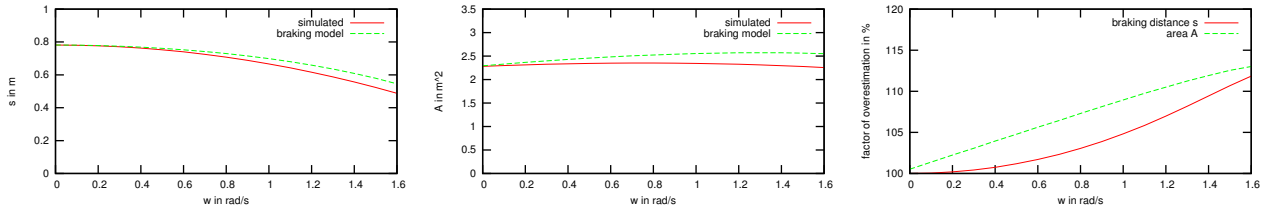


Fig. 18 Overestimation of curved braking using perfect straight forward estimation. All $(v,\omega)^T$ have $v_S = 2.5$ being one of the two braking measurements used for configuration. The plots shows how braking distance (24), area, and the overestimation factor of both depends on ω.

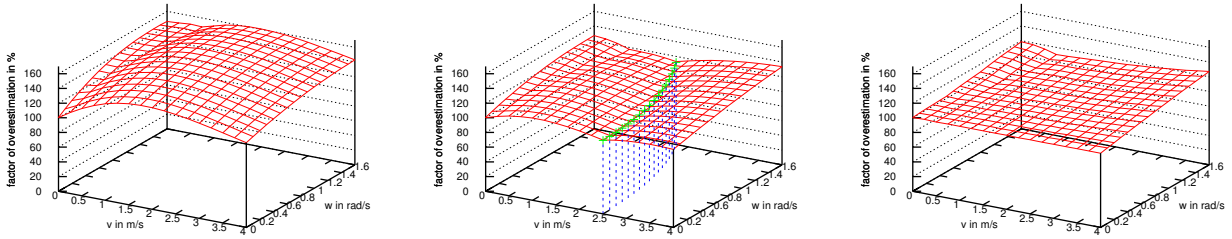


Fig. 19 Overestimation factor of the safety zone area as a function of $(v,\omega)^T$. The plot shows $\frac{A_{SAMS}}{A_{Sim}}$ for braking models configured with one, two, and nine straight braking measurements.

small (12%), and with nine negligible (1%). The percentages refer to the area of the safety zone. The factor for the braking distance s itself goes to infinity for $v \to 0$, however this is no practical problem since both true and estimated distance are small then. Based on these results we recommend using two measurements, however in industrial practice a single may suffice.

Fig. 18 indicates how much overestimation is caused by (24) bounding curved braking based on straight braking measurements in the configuration. The velocity v is chosen dependent on ω such that $v_S(v,\omega) = 2.5\text{m/s}$ is constant. This is one of the velocities measured for configuration, so the straight braking model causes no over-estimation. One can see, that even for an aggressive curve ($v = 1.85\text{m/s}, \omega = 1.5\text{rad/s}$, 1.23m radius, 2.8m/s^2 lateral acceleration) the overestimation is very moderate with 11% for distance and 13% for the safety zone area. The distance factor grows quadratic with ω because of the ω^2 in the kinetic energy (10). The area factor grows linear, not as a consequence of the overestimated distance, but because of the convex fill-in on the inside of a curve (light red area in Fig. 20-22) which is dominant. This fill-in is roughly $O(s)$ long and $O(s\alpha) = O(s\omega)$ wide and hence $O(s^2\omega)$ and linear in ω. For $\omega = 0$ the area factor is slightly (0.5%) above 100% which might be the expected value as no overestimation is introduced by the braking model in that case. This is caused by the numerical supplement mentioned in Sect. 4 which is added to the buffer radius q in (50) to cover rounding errors.

6.4 Safety Zone Computation

Fig. 20-22 show the computed safety zone for different types of motion and different numbers of measurements used to configure the braking model. It also shows the true (simulated) braking area and the area corresponding to the braking model's $(s,\alpha)^T$ but without the approximations in the safety zone computation. It can again be seen that the effect of the braking model is small to moderate. The effect of the arc approximation (35) is on the outside of the curve and too small to be visible. However, the convex fill-in on the inside (light red) is significant and could be a problem, e. g. when taking a sharp turn around a corner. This part is caused by the convex-hull representation itself not by any specific approximation. It could be avoided, if necessary, by dividing the trajectory into parts, computing a safety zone in laser scan representation for each part and then uniting the parts in laser scan representation. On the other hand, even if not completely necessary the convex fill-in automatically assures conformance with the measurement principle of our sensor, which has a straight line of sight.

Fig. 19 summarises the effects discussed so far quantitatively, showing the overestimation factor as a function of v, ω, and the number of braking measurements. For one measurement, the error in straight forward estimation is dominant visible in the bulge at $v = 2\text{m/s}$ with a maximum of 62% at $v = 1.75\text{m/s}, \omega = 1.3\text{rad/s}$. For more measurements the convex fill-in dominates, so the maximum of 34% resp. 27% is at maximum v, ω. The effect of more configuration

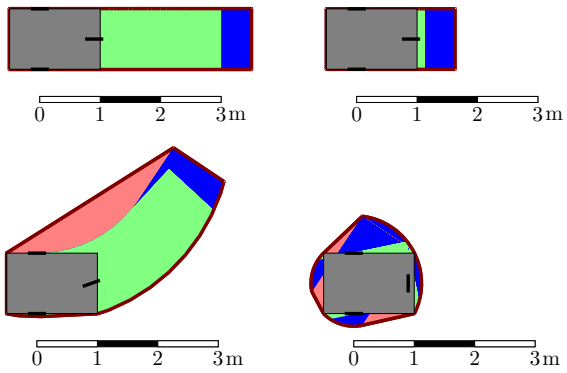


Fig. 20 Overapproximation due to braking model and safety zone computation using *one braking measurement*. The light green area is touched by the robot in simulation. The dark blue area is the same for the trajectory $(s,\alpha)^T$ given by our braking model. The overall area, i. e. the above plus the red area, is the safety zone computed by our algorithm.

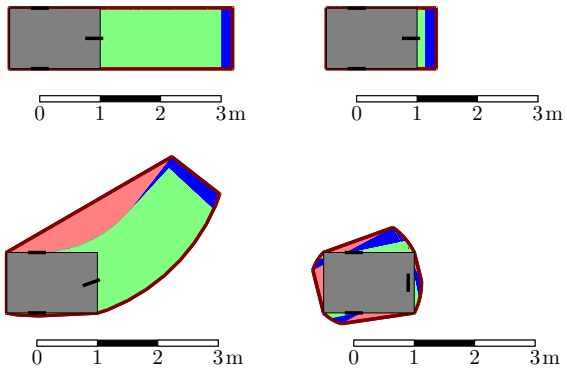


Fig. 21 Overapproximation due to braking model and safety zone computation using *two braking measurement*.

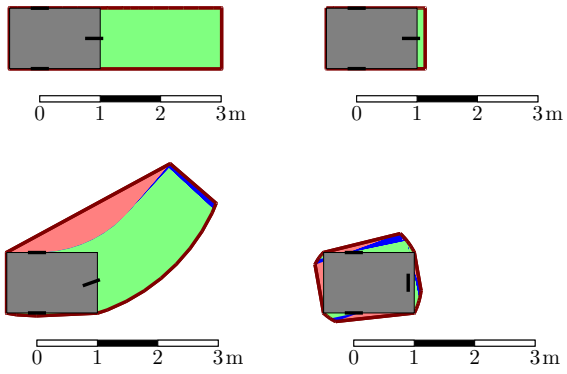


Fig. 22 Overapproximation due to braking model and safety zone computation using *nine braking measurement*.

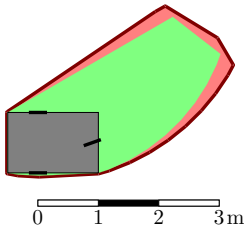


Fig. 23 Overapproximation due to v, ω interval using *2 braking measurements*. green (central area):safety zone for single braking configuration. red (overall area): full safety zone for a v, ω interval assuming 2% of error for each of the two wheel encoders.

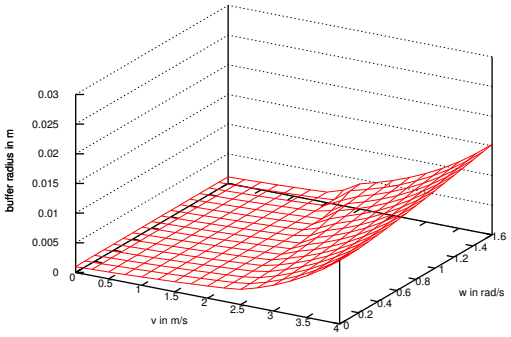


Fig. 24 Buffer radius for different $(v, \omega)^T$ intervals assuming 2% of error for each of the two wheel encoders.

measurements is small there. Nevertheless, more configuration measurements still pay off in practice, because extreme turns are only a small part of an AGV's operation.

For reference, Fig. 17 is taken from the front $(\omega = 0)$ side of the 3D plot in Fig. 19. The blue line in Fig. 19 indicates the $(v, \omega)^T$ configurations plotted in Fig. 18.

6.5 Uncertainty of Measurement

Measurements are never exact. This holds in particular for wheel encoders due to wheel slip and must be considered in a safe system. Our algorithm incorporates uncertainties by an interval $[v_{\min}, v_{\max}] \times [\omega_{\min}, \omega_{\max}]$. Fig. 23 shows how much the safety zone grows, when $\pm 2\%$ uncertainty is assumed on the two wheel encoders. The main effect is, that even a small uncertainty in the wheel velocities yields considerable uncertainty in the curvature making the safety zone grow at its stopping-end. Fig. 24 shows the buffer radius (50) bounding the non-linearity of the transition between different limiting movements in (54). With a maximum of 16mm it is rather negligible, but not by orders of magnitude.

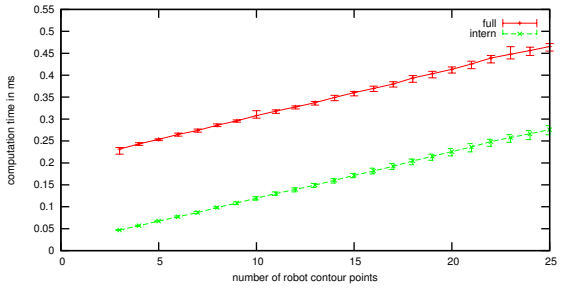


Fig. 25 Computation time on INTEL T2500@2GHz with and without conversion to a laser scan representation with a vehicle shape of n points and (v, ω intervals of $\pm 2\%$). The bars show min and max over a range of v, ω.

# of C functions (total)	39
# of C functions (formally verified)	29
Total source code size	240kB
Lines of code	5339
— program code proper	2804
— comments and specifications	2535

Table 1 Code metrics of our implementation.

6.6 Computation Time

Fig. 25 shows the computation time of a vehicle shape with n points, with and without converting to a laser scan representation. The algorithm is very fast (< 0.3ms on an Intel T2500@2GHz) for realistic robot shapes ($n < 10$). It is also very deterministic as visible in the min-max bars. Theoretically, all operations are $O(n)$ except for the $O(n \log n)$ sorting involved in Graham scan and a part of the conversion which is proportional to the number of rays in the laser scan. The first is hardly visible for such low n. The second causes an almost constant offset between both plots as the number of rays is comparably large but fixed.

7 Evaluation

Tab. 1 gives an idea about the size of the implementation described in this paper.

A common objection against formal verification is that it costs too much effort. Our actual overall effort — excluding the time needed to develop the verification framework, but including preparation and reviewing of all documents — was roughly 30 person months (p.m.). In a small survey we asked experienced project managers from industry for an assessment of the effort related to the software development according to their development model, to the verification and validation, and to the additional effort implied by an external certification. Their estimation is based on a 3-page description of the software system and personal discussions. We received two answers ranging from 12 p.m. to 87 p.m. This suggests that the effort of formal verification is at least

partly offset by time-savings due to reduced test efforts, and remains within the same ballpark as the conventional way, but offers far more confidence of correctness. Of course, no valid statistical conclusions can be drawn from such a small sample, but at least it puts our own effort into relation with industrial viewpoints.

7.1 Formal Verification in the Robotics Domain

Challenges. In practice, applying formal verification in the robotics domain faces the conflict between real-world applications involving unstructured environments and inaccurate sensors, and their idealised modeling in specifications and the formalised domain. Addressing this problem is not unique but especially important for projects applying formal verification in real world applications. Safety requires that the assumptions made in the model conform to reality.

Designing for Provability. Two concepts that proved helpful were the explicit use of intervals to accommodate for imprecision, and algorithms and representations from computational geometry. To account for imprecise measurements, our algorithm calculates safety zones for sets of velocities $[v_{\min}, v_{\max}] \times [\omega_{\min}, \omega_{\max}]$ instead of single ones. Another benefit came from representing objects by convex hulls (cf. (3)), which not only led to efficient computations, but also allowed for mathematically pleasing proofs for major parts of the algorithm. This seems to hold true for many representations and algorithms from computational geometry. By contrast, deriving approximation bounds – such as the buffer radius in (50) – often involves extensive algebraic manipulations (66), a task for which theorem provers like Isabelle are not ideally suited.

Domain. Robotics is well suited for formal verification. Formalising high-level concepts is admittedly very time-consuming. Nevertheless, much can be taken directly from textbooks so that the formalisation in Isabelle went rather smoothly. Moreover, the effort is worthwhile, as it allows simpler specifications and verification. The domain modelling is reusable for other projects, independent of a reuse of the implementation.

7.2 Specification Process

Verification as a Joint Effort. One aspect of formal verification is that because correctness relies on formal proof, it is not that crucial anymore to strictly separate the roles of tester and implementer. In contrast, the close cooperation between the verifier and the implementer boosted productivity in our case: verification became a joint effort. Writing specifications which validate the safety requirements, and can be formally verified, is not easy; it requires an understanding of the implementation, the domain model, and how the verification works. It is easy to specify something which is correct but cannot be practically verified; on the other hand, it is also a temptation to write low-level specifications which just restate what the code is doing in elementary terms without the abstraction required to state deep safety properties.

A somewhat unusual example of a close collaboration between implementer and verifier is a change of the implementation induced by verifiability considerations. The function sampling converts the buffered polygon into a sequence of vectors corresponding to a laser scan (Sec. 5.5). Initially, the specification interpreted the resulting sequence as the rays of an idealised laser rangefinder (see Fig. 5). We switched both specification and implementation to a sector-based interpretation (see Fig. 15), in which each result describes the whole area of a sector. This is more elegant because it allows us to specify the result simply as a superset of the actual braking area.

Code-Centric Specification and Verification. We experienced an interesting interplay between specification, implementation and application: at first, the specification required that if the speed of the vehicle exceeded the maximum speed for which a braking distance was measured (cf. Sec. 5), an emergency stop should be initiated. However, this turned out to be too restrictive: in typical applications, the measured maximum velocity v_m may be exceeded occasionally by a small margin, and initiating an emergency stop in these situations would severely reduce availability. Hence, the braking distance for speeds above v_m was safely overapproximated as in (5), and the specification amended accordingly.

The Importance of Being Formal. Formal specification necessitates to state requirements precisely. A beneficial side effect is that it focuses discussions and manifests design decisions. Besides the well-known issue of the ambiguities in natural language specifications, it turned out to be easier for specifiers *and* implementers to use the vocabulary of the domain formalisation to state these requirements and to reach agreement on their respective meaning. For quick sanity checks of specifications written down or modified during meetings, we provide tool support for the type-checking of specifications. This pertains both to code-related specification expressions (e. g., types of program variables) as well as Isabelle expressions used in code specifications. A typical specification meeting of 1.5 hours would end with a function specification reviewed and typechecked.

Traceability. IEC 61508 as well as other safety standards ask for traceability between adjacent phases of the development process, i. e. we have to be able to trace the realisation

of the system safety requirements down to the code. Safety requirements are formulated at a much higher abstract analysis level than the code, so usually traceability is ensured by intermediate layers in between; a definite strength of our methodology is the very strong link between the analysis level and the concrete source code. After all, we are not merely verifying algorithms on an idealised level, but work on concrete source code, as shown in Fig. 8 and 16. Since both aspects of the verification workflow are tool-supported and consistently done inside the Isabelle framework, typical leaps of faith in a safety case are avoided.

7.3 Limits of Formal Verification

The approach pursued in our project to verify software by theorem proving depends on being able to completely, mathematically specify the software's functionality. One might ask whether this approach is promising in general or whether there are problems, in particular in robotics, that do not have fully mathematically specified solutions. In the following we will discuss problems in the order of increasing severity.

Approximations. Many algorithms in robotics use approximations to speed up computation or model real world processes. There are two possibilities: Sometimes, as in our case there is a safe side of approximations, e. g. larger safety zones are allowed. Then the approximation formally becomes a bound. Safety is proven but availability in terms of the question whether the bound is tight enough is not proven, because it is practically important but not safety-critical.

If there is no safe side, a bound on the approximation error must be proven, such as the buffer radius in (50). This is often quite some effort but also increases the confidence in the approximation. Overall it is our impression that for most approximations it is worth to try proving a hard bound on the error.

Optimisations. Many robotics algorithms follow the paradigm of optimising a cost function. First, optimal does not mean good. Second, many optimisation algorithms only find local minima. Assuming it to be global is most often a leap of faith. Overall, proving meaningful conditions on the result of an optimisation algorithm is a hard scientific challenge, even though it provides a deeper insight.

A solution is to reverse the optimisation problem and implement a check of the quality of the optimisation output, which switches to a safe mode if the output is not good enough. This is attractive, because the optimisation itself need not be verified at all, only the subroutine that checks the result, which usually is much easier.

Monte Carlo Algorithms. Randomised algorithms have proven powerful tools for many robotics problems. At first sight, randomisation fits well to the approach in IEC 61508 to specify an upper bound for the probability of failure. However, we are not aware of concrete bounds as needed for a certification, only asymptotic statements assuring that with infinite effort the probability of failure goes to 0.

The Lower Limit. First, if the domain is sufficiently simple it is preferable to apply automatic rather than interactive verification methods. Examples of such domains have been outlined in Sec. 4. These domains cover a large fraction of industrial safety applications. Employing formal verification demands an exact and complete description of the model, experience in mathematics and availability of the verification tools. This entails practically relevant efforts that must be weighed up against the advantage of having a formally proved algorithm.

Heuristics. Finally, many successful algorithms in robotics are partially heuristic. Sometimes the heuristic just guides a search process without affecting correctness, e.g. heuristics for A* search; in this case, formal verification is still possible. However, often the motivation for a heuristical approach is to avoid a tedious theoretical analysis of the problem to be solved. In that case, proving correctness may even be theoretically possible but would require exactly the theoretical analysis avoided before.

Also many detection algorithms in computer vision belong to this category, e. g. face detection, because it is simply impossible to fully formalise how a face looks in an image under various conditions. In these cases no provable result about the overall functionality can be obtained and an empirical approach is more adequate. Nevertheless, subalgorithms, e. g. computation of a filter, could still be specified and verified.

8 Closing Remarks

In this paper, we have presented a safety function for autonomous robots which has been certified for use in applications up to SIL 3. The safety function computes a safety zone for the moving robot, which can be safeguarded by a laser rangefinder. The safety zone is a safe, conservative and yet small enough to be practically usable approximation of the area covered by the robot while braking down to a standstill from the current velocity. The development conformed to the relevant industrial safety standard, IEC 61508, which was certified to us by an external certification authority. To this end, most of the implementation has been formally verified with respect to the safety requirements. The main means of verification was a tool based on the theorem prover Isabelle.

The SAMS safety module is an example of a complex, realistic, industrial-strength safety application for robotics. Our experiences have shown:

(i) Safety concerns all phases of the development process; it cannot be added on as a feature after the fact. In other words, we should not expect to take an arbitrary existing algorithm and 'make it safe'.
(ii) Formal verification with higher-order logic and computer-aided theorem proving is well suited to certify the safety of an implementation of a complex robotics algorithm.
(iii) The effort of mathematically analysing an algorithm to the extent needed for that provides a deeper understanding of the algorithm and is a value also outside the certification context.

The focus of safety certification is not correctness in a "philosophical" sense, but to prevent errors that have led to accidents in the past. In so far, it is well accepted to cover some safety properties by formal verification and use other methods (tests, code reviews, etc.) for the rest. Also, even if the whole problem is too complex, formally verifying certain aspects can be fruitful. The safety module guarantees freedom of collision with static obstacles. This is standard industry practice; the underlying assumption is that in work environment, no human that can move willfully steps into the path of an oncoming AGV, so we mainly need to safeguard against collision with incapacitated or unaware humans. Note that even so the safety function will provide a degree of safety, as it will still reduce the impact speed if colliding with a moving obstacle. Extending our algorithm to dynamic obstacles is possible, but there needs to be a maximum obstacle speed (otherwise availability will suffer), and that needs to be given by an industrial or domestic safety standard.

In closing, we can positively state that safety considerations *can* be taken into account in robotics in a manner conforming with legal requirements and existing standards, but safety is more than a quick code review on a Friday afternoon. It needs a thorough analysis of the safety requirements and the algorithms, a meticulous verification process, and external reviewing. With this in mind, robots can *safely* share the human living and working spaces.

A Covered Verification Measures

Annexes A and B of IEC 61508-3 give guidance on the selection of procedures and measures to fulfill the requirements of the different life cycle phases defined in the standard. The latter are presented in tabular form and pertain to the software safety requirements specification, software design and development, integration, modification as well as verification and validation.

Being a basic safety standard applicable in various domains, there is little concrete advice about the content of functional requirements. Instead, the focus is on the structure of the relevant documents. The requirements about programming practice and code analysis are more informative; those covered by the application of our methodology are shown in Tab. 2.

Table A.4	Software design and development: Detailed design	
	1c	Formal methods such as, e. g., CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM, and Z
	2	Computer-aided design tools
	5	Design and coding guidelines (detailed in B.1)
	6	Structured programming
Table A.9	Software verification	
	1	Formal proof
	3	Static analysis (detailed in B.8)
Table B.1	Design and coding guidelines	
	1	Use of coding guidelines
	2	No dynamic objects
	3a	No dynamic variables
	4	Restricted use of interrupts
	5	Restricted use of pointers
	6	Restricted use of recursions
	7	No unconditional jumps in programs written in higher-level languages
Table B.8	Static analysis	
	1	Marginal value analysis
	3	Control flow analysis
	4	Data flow analysis
	5	Error estimation
	8	Symbolic execution

Table 2 Measures as required by IEC 61508-3, Annex A and B, and covered by the use of formal software verification.

References

Althoff M, Althoff D, Wollherr D, Buss M (2010) Safety verification of autonomous vehicles for coordinated evasive maneuvers. In: Intelligent Vehicles Symposium (IV), 2010 IEEE, pp 1078–1083

ANSI R15.06 (1999) Industrial robots and robot systems – safety requirements

Barnes J (2003) High Integrity Software: The SPARK Approach to Safety and Security. Addison-Wesley

Baudin P, Cuoq P, Filliâtre JC, Marché C, Monate B, Moy Y, Prevosto V (2008) ACSL: ANSI C specification language. `http://frama-c.com/download/acsl_1.4.pdf`, version 1.4, retrieved Jan 2011

Bengtsson J, Larsen KG, Larsson F, Pettersson P, Yi W (1995) UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In: Proc. of Workshop on Verification and Control of Hybrid Systems III, Springer, no. 1066 in Lecture Notes in Computer Science, pp 232–243

Bensalem S, Gallien M, Ingrand F, Kahloul I, Nguyen TH (2009) Designing autonomous robots: Toward a more dependable software architecture. IEEE Robotics & Automation Magazine 16(1):67–77

Bensalem S, Bozga M, Nguyen TH, Sifakis J (2010a) Compositional verification for component-based systems and application. Software, IET 4(3):181 –193, DOI: 10.1049/iet-sen.2009.0011

Bensalem S, da Silva L, Gallien M, Ingrand F, Yan R (2010b) Verifiable and correct-by-construction controller for robots in human environments. In: DRHE 2010 Dependable Robots in Human En-

vironments, Seventh IARP Workshop on Technical Challenges for Dependable Robots in Human Environments, Toulouse, France
Braman JMB (2009) Safety verification of goal-based control programs for autonomous robotic systems. Talk at ICRA09 workshop on formal methods in robotics
Braman JMB, Murray RM, Wagner DA (2007) Safety verification of a fault tolerant reconfigurable autonomous goal-based robotic control system. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, San Diego, pp 853–858
Burdy L, Cheon Y, Cok D, Ernst MD, Kiniry J, Leavens GT, Leino KRM, Poll E (2005) An overview of JML tools and applications. International Journal on Software Tools for Technology Transfer 7(3):212– 232
Cohen E, Dahlweid M, Hillebrand M, Leinenbach D, Moskal M, Santen T, Schulte W, Tobies S (2009) VCC: A practical system for verifying concurrent C. In: TPHOLs 2009, Springer, Lecture Notes in Computer Science, vol 5674
Dion B, Gartner J (2005) Efficient development of embedded automotive software with IEC 61508 objectives using SCADE drive. In: VDI 12th International Conference: Electronic Systems for Vehicles, VDI, pp 1427–1436
Dolginova E, Lynch N (1997) Safety verification for automated platoon maneuvers: A case study. In: Hybrid and Real-Time Systems, Springer, Lecture Notes in Computer Science, vol 1201, pp 154–170
Du Toit NE, Wongpiromsarn T, Burdick JW, Murray RM (2008) Situational reasoning for road driving in an urban environment. In: Intelligent vehicle control systems : proceedings of the 2nd International Workshop on Intelligent Vehicle Control Systems (IVCS), INSTICC Press, pp 30–39
EN 1525 (1997) Safety of industrial trucks – driverless trucks and their systems
Ericson C (2005) Real-Time Collision Detection. Morgan Kaufmann
European Parliament and Council (2006) Directive 2006/42/EC. Official Journal of the European Union L 157
Fiorini P, Shillert Z (1998) Motion planning in dynamic environments using velocity obstacles. International Journal of Robotics Research 17:760–772
Fox D, Burgard W, Thrun S (1997) The dynamic window approach to collision avoidance. IEEE Robotics and Automation Magazine 4(1):23–33
Fraichard T (2007) A short paper about motion safety. In: Proceedings of the IEEE International Conference on Robotics and Automation
Fraichard T, Asama H (2004) Inevitable collision states — a step towards safer robots? Advanced Robotics 18(10):1001–1024
Gates B (2007) A robot in every home. Scientific American 296:58–65, DOI: 10.1038/scientificamerican0107-58
Gordon M, Milner R, Wadsworth C (1979) Edinburgh LCF: a Mechanised Logic of Computation, Lecture Notes in Computer Science, vol 78. Springer
Gurevich Y (2000) Sequential abstract state machines capture sequential algorithms. ACM Transactions on Computational Logic 1(1):77– 111
Harel D (1987) StateCharts: a visual formalism for complex systems. Science of Computer Programming 8:231– 274
Henzinger TA, Ho PH, Wong-Toi H (1997) HyTech: A model checker for hybrid systems. Software Tools for Technology Transfer 1:110–122
Hoare CAR (2009) Viewpoint: Retrospective: An axiomatic basis for computer programming. Commun ACM 52(10):30–32
Holzmann GJ (2003) The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley
IEC 61508 (2000) Functional safety of electrical/electronic/-programmable electronic safety-related systems
ISO 10218-1 (2006) Robots for industrial environments – safety requirements – part 1: Robot
ISO 15623 (2002) Road vehicles : Forward vehicle collision warning system - performance requirements and test procedures
ISO 26262 (2011) Road vehicles - functional safety. (final draft)
Khatib O (1986) Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. The International Journal of Robotics Research 5(1):90–98
Lankenau A, Röfer T (2001) A safe and versatile mobility assistant. IEEE Robotics and Automation Magazine 8(1):29–37
Loos S, Platzer A, Nistor L (2011) Adaptive cruise control: Hybrid, distributed, and now formally verified. In: Formal Methods, Springer, Lecture Notes in Computer Science, vol 6664, pp 42–56
Lüth C, Walter D (2009) Certifiable specification and verification of C programs. In: Formal Methods, Springer, Lecture Notes in Computer Science, vol 5850, pp 419–434
Meikle L, Fleuriot J (2009) Mechanical theorem proving in computational geometry. In: Automated Deduction in Geometry, Springer, Lecture Notes in Computer Science, vol 3763
Meyer B (1991) Design by contract. In: Mandrioli D, Meyer B (eds) Advances in Object-Oriented Software Engineering, Prentice Hall, pp 1– 50
Minguez J, Montano L (2004) Nearness diagram (ND) navigation: Collision avoidance in troublesome scenarios. IEEE Transactions on Robotics and Automation 20(1):45–59
MISRA (2004) MISRA-C:2004 – Guidelines for the use of the C language in critical systems. Motor Industry Research Association (MIRA) Limited, Nuneaton, UK
Nilsson NJ (1984) Shakey the robot. Tech. Rep. 323, SRI International, Menlo Park, California
Nipkow T, Paulson LC, Wenzel M (2002) Isabelle/HOL — A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, vol 2283. Springer
Parthasarathi R, Fraichard T (2007) An inevitable collision state-checker for a car-like vehicle. In: Proceedings of the IEEE International Conference on Robotics and Automation
Philippsen R, Siegwart R (2003) Smooth and efficient obstacle avoidance for a tour guide robot. In: Proceedings of the IEEE International Conference on Robotics and Automation
Platzer A, Clarke EM (2009) Formal verification of curved flight collision avoidance maneuvers: A case study. In: Formal Methods, Springer, Lecture Notes in Computer Science, vol 5850, pp 547–562
Puri A, Varaiya P (1995) Driving safely in smart cars. In: Proceedings of the American Control Conference, vol 5, pp 3597–3599
Rabe C, Franke U, Gehrig S (2007) Fast detection of moving objects in complex scenarios. In: Proceedings of the IEEE Intelligent Vehicles Symposium, pp 398–403
Roscoe AW (1998) The theory and practice of concurrency. Prentice Hall
RT-Tester (2006) User Manual. Verified Systems International GmbH, `http://www.verified.de/en/products/rt-tester`, retrieved Jan 2011
Schlegel C (1998) Fast local obstacle avoidance under kinematic and dynamic constraints for a mobile robot. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems
Simon D, Pissard-Gibollet R, Arias S (2006) Orccad, a framework for safe robot control design and implementation. In: 1st National Workshop on Control Architectures of Robots : software approaches and issues CAR'06, Montpellier, France
Täubig H, Bäuml B, Frese U (2011) Real-time swept volume and distance computation for self collision detection. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), San Francisco, CA, United States
Varaiya P (1993) Smart cars on smart roads: problems of control. IEEE Transactions on Automatic Control 38(2):195–207

Victorino AC, Rives P, Borelly JJ (2003) Safe navigation for indoor mobile robots. part I: A sensor-based navigation framework. International Journal of Robotics Research 22(12):1005–1118

Winner H, Haskuli S, Wolf G (eds) (2009) Handbuch Fahrerassistenzsysteme. Vieweg+Teubner, (German)

Wongpiromsarn T, Mitra S, Murray R, Lamperski A (2009) Periodically controlled hybrid systems. In: Hybrid Systems: Computation and Control, Springer, Lecture Notes in Computer Science, vol 5469, pp 396–410