Synthesis of safety rules for active monitoring: application to an airport light measurement robot

Lola Masson, Jérémie Guiochet, Hélène Waeselynck
LAAS - CNRS
Toulouse, France
firstname.lastname@laas.fr

Augustin Desfosses, Marc Laval
Sterela
Toulouse, France
firstname.lastname@sterela.fr

***Abstract*—Safety-critical autonomous systems, like robots working in collaboration with humans, are about to be used in diverse environments such as industry but also public spaces or hospitals. Those systems evolve in complex and dynamic environments and are exposed to a wide variety of hazards. Several techniques may be used to ensure that their misbehavior cannot cause unacceptable damage or harm. One of them is active safety monitoring. A safety monitor is a component responsible for maintaining the system in a safe state despite the occurrence of hazardous situations. In this paper, we study the introduction of safety monitoring into an airport light measurement robot. The specification of the monitor follows a principled approach that starts with a hazard analysis and ends with a set of safety rules synthesized based on formal methods. This study illustrates the benefits of the approach, and shows the impact of safety on the development of an autonomous system.**

I. INTRODUCTION

The autonomous operation of mobile robots in unstructured and human-shared environments is subject to hard safety constraints. Indeed, the misbehavior of such systems could cause unacceptable damage or harm. This concern is one of the major obstacles to the effective deployment of otherwise technically feasible robots in many real-life situations. As an example, this paper concerns an autonomous robotic system for airfield lighting system maintenance, which is being developed by Sterela. This robot should automate some tedious light measurement tasks that are done by human operators. But today, no autonomous system is allowed on runways due to safety issues. So, whatever the innovative features of the robot, and however useful the provided service may be, there will be no market for it unless Sterela provides a solid safety case to the airport authorities. As a step toward building such a case, we study the specification of a solution that adds safety monitoring to the system.

A safety monitor is a device responsible for safety only, and kept independent from the main control channel. It is intended to act as the ultimate protection against interaction faults or arbitrary behavior of the control channel that adversely affect safety. To serve its protection role, it is equipped with means for context observation (i.e., sensors) and has the ability to trigger safety interventions when a potentially dangerous situation is detected. The monitor behavior is specified by *safety rules* of the form: if *condition* then *intervention*. While ensuring safety, the rules should still permit functionality of the system, a property that we call *permissiveness*.

Previous work has proposed a systematic process to produce the safety rules [1]. It starts by a hazard analysis method, HAZOP-UML [2], from which a set of safety invariants is derived. The monitor is in charge of maintaining the system in states satisfying these invariants. After a formalization step, appropriate safety rules are synthesized by SMOF (Safety MOnitoring Framework) [3], [4], a framework that uses the NuSMV model checker. SMOF accommodates both safety and permissiveness requirements, which we express in terms of state reachability properties.

The paper reports on the application of this process to the light measurement robot. Section II presents the characteristics of the robot. Section III explains the HAZOP-UML method and gives its outcomes for this specific case study. Section IV illustrates the synthesis of safety rules on two examples of invariants derived from HAZOP-UML. After an overview of related work on Section V, we conclude on the benefits of the proposed approach, and on the impact of safety on the development of the studied robot.

II. INDUSTRIAL CASE STUDY

The studied use case concerns maintenance and control of the lights along the airport runways. The International Civil Aviation Organization (ICAO) recommends monthly measurement of the light intensity of airfield lighting installations, using a certified device. If the light intensity does not comply with the ICAO requirements, no air traffic can be allowed on the airfield.

Currently, human operators perform the measurements but this is a burdensome and displeasing task. It has to be done late at night, typically from 1am to 4am. The repeated exposure to intense lights in surrounding dark may cause eye strain. The task would thus be a perfect fit for robotic operation.

The French company Sterela is developing a prototype system to serve this purpose. The robot consists of a mobile platform and a commutable payload (Fig. 1). The platform, called 4MOB, is a four wheel drive vehicle. The payload is a certified photometric sensor on a support deported on the side of the platform. The deported sensor moves above the lights (15-20cm) with a maximum speed of 1.4m/s. A human operator is supervising the mission with a digital tablet from the extremity of the runway. As the robot operates at night

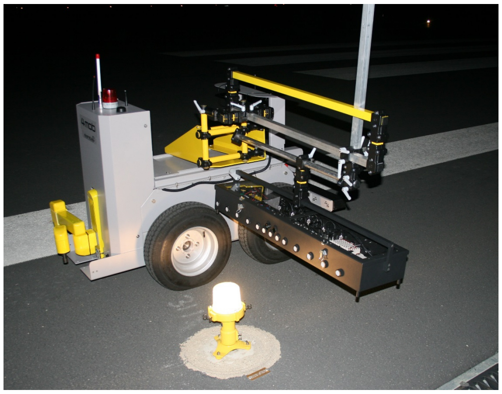

Fig. 1. Sterela robot with sensor support

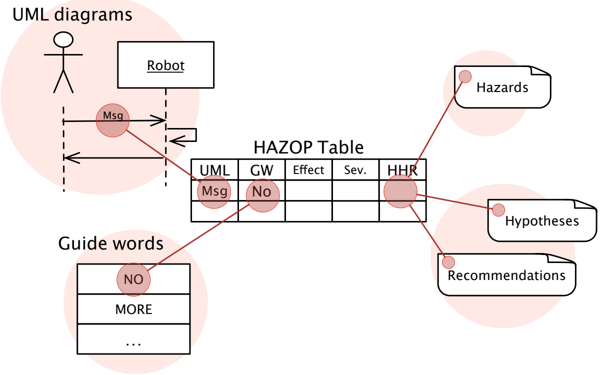

Fig. 2. HAZOP-UML overview

and at long distances, the operator has no direct visual contact with it. Safety concerns are the main barrier to the deployment of such an autonomous robot on real airports. Other operators and service vehicles could be present on the runway on which the robot evolves, and it must not constitute a hazard to them. Moreover, even if the robot could be allowed on runways, there would be airfield areas strictly forbidden to it: it must be ensured that the robot never traverses them. Finally, the air traffic controllers should keep the ability to re-open a runway as quickly as possible in case of a landing emergency. In that case, Sterela originally considered an automated runway evacuation procedure according to which the robot aborts its mission and rushes into a safe area.

In order to better identify the risks induced by the robot operation, and the potential solutions to address them, Sterela and LAAS applied two safety-related methods. The first one, HAZOP-UML [5], is a hazard analysis method. Based on its outcomes, the second method, SMOF (Safety MOnitoring Framework [6]), determines the safety rules to be implemented by a safety monitor. The methods were applied at an early stage where some design decisions are still open and can be tuned to accommodate safety concerns.

III. HAZARD ANALYSIS WITH HAZOP-UML

The first step of the process is to identify a list of hazardous situations prior to their formalization as safety invariants. This study has been done using a model-based hazard analysis technique developed at LAAS called HAZOP-UML [2]. It combines the well known modeling language UML (Unified Modeling Language) and a hazard analysis technique called HAZOP (HAZard OPerability) originally used in petrochemical industry. As presented in Fig. 2, based on the UML models and a bunch of predefined guidewords, a systematic deviation analysis is performed. It results in a list of hazardous situations that might occur during system operation. An extract of the produced HAZOP tables is given in Table I. Overall, a total of 1623 deviations were analyzed as presented in Table II.

	4MOB
Use cases	5
Conditions (pre,post,inv)	24
Sequence diagrams	5
Messages	47
Deviations	1623
Interpreted deviations	226
Interpreted deviations with severity > 0	97
Number of hazards	10

TABLE II
STATISTICS FOR THE APPLICATION OF HAZOP-UML

Being a systematic approach, HAZOP-UML typically induces the analysis of many potential deviations. A first way to control complexity is by setting the level of detail of the UML models. This has been done through the number of UML elements (47 messages and 24 conditions). A second way is to provide tool support that alleviates the effort. For this, Sterela has extended their requirement editing tool to partially generate UML diagrams and HAZOP tables. This tool is connected to their application lifecycle management software based on the open source Tuleap platform [7].

A first result of such a study is a list of recommendations for the use or the design of the robot. For instance, an automated runway evacuation function was originally planned in case of emergency landing. However, it became clear that safety cannot depend on such a complex function. It involves the whole chain of control including sensors and actuators, the power supply, the navigation software, and every one of those would be assigned a high integrity level. The analysis highlights potential deviations for these components as in Table I for the battery: an incomplete charge of the battery could compromise the success of the evacuation. But commercially available batteries do not provide a high integrity level for the battery charge rate information. Ensuring a high integrity navigation software service would obviously be even more difficult. The automated evacuation function was then removed and replaced by a manual evacuation procedure, with a high integrity on the last known localization of the robot.

A second result is the identification of the hazards to be treated by the safety monitor. To prepare the synthesis of safety rules, the hazards are reformulated as the violation of

Entity	Attribute type	Guide word	Deviation	Effect	Severity	Safety recommendation
[4mob battery is being charged]	Pre-condition	Part of	The battery is partially charged	Not enough battery during the mission: problem to evacuate the runway in case of emergency	High	Check the battery rate at any time and determine minimum value to be ready to evacuate

TABLE I
HAZOP TABLE EXTRACT

invariants based on observable variables. Not all the identified hazards can be reformulated this way. Hence, a subset of hazards are in the scope of the monitor. For the studied robot, the monitor may address 5 hazards (out of 10):

1) Collision with an obstacle (including lights);
2) Fall of the robot during loading/unloading - the robot can fall on the ground or on an operator;
3) Movement in an unauthorized zone - some areas of the airport are not allowed to the robot;
4) Inability for the operator to locate the robot;
5) Dangerous velocity - the robot can go too fast in terms of linear or angular velocity.

Among the 5 remaining hazards, one has been eliminated by giving up the corresponding functionality (automated runway evacuation); and the others will be managed by operating procedures (availability of the operator in case of emergency for example).

HAZOP-UML provides a systematic study of hazards. Its application requires resources and time, but the development of a dedicated tool, connected to a requirement tool, reduces the effort to complete the analysis. HAZOP is particularly well adapted to the context of safety monitoring, since it focuses on operational hazards (other hazards such as electric shocks, sharping edges, etc. are not studied). The next section presents how to infer safety rules from those operational hazards.

IV. FORMAL MODELING AND SYNTHESIS OF SAFETY RULES

This section illustrates the synthesis of safety rules with SMOF for two invariants of growing complexity, after an overview of the basic concepts.

A. Basic concepts

The monitor is responsible for enforcing the safety invariants determined by the hazard analysis. If the system violates one of these invariants, it enters what we call a catastrophic state and we assume it to be irreversible. Figure 3 gives a schematic view of the system state space observed from the perspective of the monitor. Each safety invariant SI determines a partition into catastrophic and non-catastrophic states. The non-catastrophic states can in turn be partitioned into safe and warning states, in such a way that any path from a safe state to a catastrophic one traverses a warning state. The warning states correspond to safety margins on the values of observable variables. For example, let us assume that the invariant involves a predicate $v < V_{max}$. The monitor may detect a warning state when v reaches the value $V_{max} - margin$. It may then decide to trigger interventions to prevent any evolution toward the catastrophe. This is done according to safety rules determining which interventions to trigger in the various warning states. The set of rules form a strategy that must fulfill two properties:

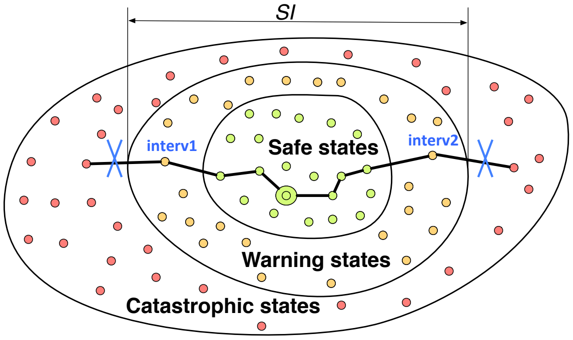

Fig. 3. System state space from the perspective of the monitor

Safety: non reachability of the catastrophic states;

Permissiveness: reachability of every non catastrophic state. The permissiveness property is necessary to avoid safe strategies that would constrain system behavior to the point of compromising its functionality (e.g., always engaging brakes to forbid any movement). We require the monitored system to keep its ability to reach a wide range of states. The user can select two levels of permissiveness: simple permissiveness i.e. every non catastrophic state is reachable from the initial state; universal permissiveness i.e. every non catastrophic state is reachable from every other non catastrophic state. The universal permissiveness is used by default because it is the less restrictive. If no strategy is found, the user can select simple permissiveness, or tune the level of permissiveness to the states under consideration.

SMOF provides a template to formalize the various elements decribed so far: the behavior model with a partition into safe, warning and catastrophic states; the available interventions modeled by their effect on observable state variables, possibly under some preconditions; the safety and permissiveness properties. The template offers predefined modules, as well as auto-completion facilities. For example, the template contains a generic $LiveProp$ module for permissiveness properties, and its instantiation for all non-catastrophic states is automatically done. Likewise, the specification of the behavior model and interventions is much simplified by entering parameters to declaration modules, and letting SMOF automatically generate the formal expression of warning states as well as the glue for associating interventions with warning states. The formalization language is the one of the NuSMV model checker [8].

After this formalization step, the synthesis tool uses a branch-and-bound algorithm to explore intervention strategies. During exploration, it calls NuSMV to assess the adequacy of

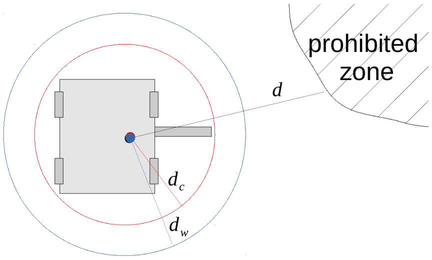

Fig. 4. The robot must not enter a prohibited area

Position of the robot relative to the prohibited zone	**Real distance interval**	**Discrete variable**
The robot is too close to the prohibited zone	$d \leq d_c$	$d = 0$
The robot is close to the prohibited area	$d_c < d \leq d_w$	$d = 1$
The robot is far away from the prohibited area	$d \geq d_w$	$d = 2$

TABLE III
PARTITIONING OF THE VARIABLE d

the candidate rules with respect to safety and permissiveness. Inadequate subsets of rules, which cannot be part of a complete safe and permissive solution, are discarded. The tool returns a list of alternative sets of rules, each forming an adequate strategy for the given invariant to enforce.

The formalization and rule synthesis is done for each invariant separetely. Then a last step is to merge the models and to check for the consistency of the strategies retained for the different invariants.

This method is illustrated in the next sections, with a zoom on two invariants.

B. The robot must not enter a prohibited area

We intentionally choose here a simple invariant to illustrate the method and the tool.

The monitor needs only one observation, d: distance to the prohibited area (see Fig. 4). The catastrophic state is when $d \leq d_c$. In order to detect that the distance gets close to d_c, we define a warning threshold d_w.

The distance values, from the perspective of the monitor, are thus partitioned into three classes: the robot is far away from the area, close to it, or too close (catastrophic states). In the NuSMV model, this partition is simply encoded by introducing an abstract variable d with three values $(0, 1, 2)$ (see Table III).

The declaration of the variable uses the predefined `continuity` module of the SMOF modeling template. Its parameters determine the range of values of the abstract variable, as well as its initial value:

```
--Continuity(lower bound, upper bound, initial
value)
d: Continuity(0,2,2);
```

As its name suggests, the module also encapsulates a continuity constraint on the evolution of values. The next value can only remain the same, or be incremented or decremented by one:

```
next(d) = d | next(d) = d+1 | next(v) = d-1;
```

In particular, the distance cannot directly jump from 2 (far away) to 0 (too close). Of course, the justification of this constraint will not come for free: the use of the continuity module puts an explicit requirement on the implementation of the observation. The chosen threshold and sampling rate must ensure that, even in the worst case velocity of the robot, the monitor will always see an intermediate value in the range $[d_w, d_c[$ before d_c is reached.

The definition of catastrophic states uses the keyword `cata`: `DEFINE cata := d=0`. In the template, the safety property is predefined as `!cata`, and there is a constraint stating that a catastrophe is irreversible.

Finally, candidate interventions have to be declared. This is done by instantiating the predefined `Interv` module with parameters describing the effect of the intervention on observable variables, under some preconditions. Two types of preconditions are considered: a static precondition specifies a condition for states in which the intervention can be triggered; a sequential precondition specifies a condition on the previous state before entering the triggering state. The module encapsulates the constraint that the effect is guaranteed only if the preconditions hold. For example, consider the following declaration of an intervention that brakes until the platform is stationary (full stop):

```
--Interv(static precondition, sequential
precondition, effect)
full_stop: Interv(TRUE, d=2, next(d)=d);
```

The intervention is allowed in any state (static precondition is `TRUE`), but the effect is guaranteed only if the previous distance was far away from the prohibited zone (`d=2`, i.e., the warning threshold has just been crossed). The declaration must be seen as an implementation requirement for intervention means, to be justified by appropriate calculation of the threshold based on the worst case braking distance.

For this simple invariant, the modeling task merely consists of entering the three lines we have just described: the declaration of the state variable, of the cata predicate and of the intervention. In particular, the SMOF tool automatically identifies the warning states, i.e., states having a one-step transition to the catastrophic ones. Here, the warning states are trivially `d=1`. The tool also automatically instantiates the state reachability properties for permissiveness.

The rule synthesis algorithm explores associations of interventions with warning states. Since the model has one warning state and one intervention, there is a single possible strategy: it triggers the full stop if `d=1`. Still, the tool returns no solution with the default universal permissiveness property. One has to switch to simple permissiveness to get the expected strategy. Indeed, the full stop freezes the distance to the prohibited area: its trigger `d=1` remains true forever. The robot is blocked at its current location, requiring an action of the operator.

Another option would be to use a more sophisticated intervention, which would prevent the robot from going closer to the prohibited area but would allow moves to get away from it. This intervention could be modelled as follows: `restrict_moves: Interv(TRUE, d=2, next(d)!=0);` and would ensure universal permissiveness. But this is a matter of compromise, as the specification of `restrict_moves` is much more difficult to implement than the one of the full stop.

This simple but real-life example allowed us to illustrate the use of the SMOF template, as well as the exploration of alternative strategies. Although the modeling is here trivial, it forces the user to make the implementation requirements explicit, which is a valuable outcome. The next invariant is more challenging with respect to both the modeling and the synthesis of rules.

C. The robot must not collide with an obstacle

The absence of collision is an interesting problem for the studied robot. The robot should be allowed to move very close to the lights, so that its deported sensor passes above them. Still, it must not collide with them. Also, it must not be allowed to move close to other types of obstacles like humans or service vehicles.

The formalization of the problem was done in two steps. We first considered a simple case, with a single idealized obstacle modeled by a pair of (x, y) coordinates. It helped us to identify some assumptions on the obstacles to address, as well as some requirements on how to observe the neighborhood of the robot and which interventions to provide. The second model was more complex, accommodating multiple obstacles and their spatial extension. It allowed us to confirm the strategy synthesized from the simple case.

1) Simple case: We assume that the robot is not responsible for rear-end collisions (like in car accidents). It is then sufficient to monitor the neighbourhood in front of, and at the sides of the robot. However, the robot must not be allowed to move backwards, or to make sharp turns (remember it is with four-wheel steering, it might turn 360 ° on the spot). Such movements would be dangerous for stationary obstacles close behind it. To avoid this, we add two invariants (not presented here) that restrict the direction and curvature radius of movement.

We consider two types of obstacles: high obstacles, including humans, which are too tall to pass under the deported sensor, and low obstacles (20 cm max), which can pass under it. The robot must be able to make the difference, at least for obstacles that are located at the same side as the deported sensor. A solution may be to equip the robot with 2D Lidar units, some on the platform and some on the deported extension. Fig. 5 shows an exemplary implementation. Please note that the figure is for illustrative purposes and does not represent a real implementation. The low obstacles, including lights, are assumed stationary: otherwise, there would be no safe strategy allowing the robot to move close to them.

The monitor can use the following observations:

x : abscissa of the obstacle in the robot's referential
y : ordinate of the obstacle in the robot's referential
v : robot's velocity

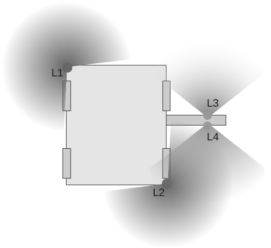

Fig. 5. Disposition of the lasers on the robot

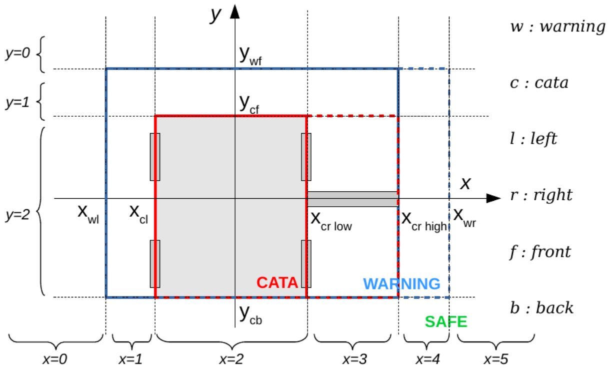

Fig. 6. Visual representation of the classes of (x,y) coordinates

For x		**For y**	
$x < x_{wl}$	$x = 0$	$y > y_{wf}$	$y = 0$
$x_{wl} \leq x < x_{cl}$	$x = 1$	$y_{cf} < y \leq y_{wf}$	$y = 1$
$x_{cl} \leq x \leq x_{crlow}$	$x = 2$	$y_{cf} \leq y \leq y_{cb}$	$y = 2$
$x_{crlow} < x \leq x_{crhigh}$	$x = 3$		
$x_{crhigh} < x \leq x_{wr}$	$x = 4$		
$x > x_{wr}$	$x = 5$		
For v		**For the $type$ (when $x > 2$)**	
The robot is standstill	$v = 0$	The obstacle is low	$type = 0$
The robot is moving	$v = 1$	The obstacle is high	$type = 1$

TABLE IV
PARTITIONING AND ABSTRACTION OF THE VARIABLES FOR THE SIMPLE CASE MODEL

$type$: type of the obstacle (high or low)

Their values are partitioned into classes and abstract variables are defined to encode them. Fig. 6 gives a visual representation of how the space around the robot is partitioned, and Table IV recaps the encoding for all variables.

The (x, y) coordinates are declared with continuity constraints. An additional constraint models the stationary assumption of low obstacles (having $type = 0$). However, the type of obstacle needs to be observed only on the right side of the robot ($x > 2$). In other location areas, the type is forced to the default value 1 (the obstacle is potentially high and mobile).

The catastrophic states are collisions with the platform (for any type of obstacle) or with the deported sensor (for a high obstacle) at a non-zero velocity. Using the encoding into abstract variables, `cata` is defined as:

```
v=1 & ((x=2 & y=2) | (type=1 & x=3 & y=2))
```

The full stop intervention is redefined for these variables. It stops the platform under a threshold crossing precondition, i.e., the obstacle was previously far away on the left ($x = 0$), or on the right ($x = 5$) or in front ($y = 0$) of the robot. The declaration is as follows:

```
full_stop: Interv(TRUE, x=0 | x=5 | y=0,
next(v)=0);
```

There is no permissive strategy if this intervention is the only one available. To ensure safety, the robot stops whenever any type of obstacle enters the warning zone. But the light measurement functionality requires the robot to move close to a low obstacle (i.e., $v = 1$ & $x = 3$).

A second intervention is then added: restrict the move of the robot in direction to the right. The effect works for stationary obstacles only, that is, for low obstacles only under our modeling assumption.

```
restrict_right_curve: Interv(type=0, TRUE,
next(x)!=x-1)
```

With the two above interventions, the rule synthesis returns one safe and permissive solution:

- Brake when a high obstacle is close, irrespective of whether the robot is moving or not:
  ```
  trigger_full_stop: type=1 &
  (x=1 | x=2 | x=3 | x=4) & (y=1 | y=2);
  ```
- Restrict the curve when a low obstacle is in the deported sensor zone and the robot is moving:
  ```
  trigger_restrict_right_curve: type=0 & x=3
  & (y=1 | y=2);
  ```

The synthesis took 3.2s on an Intel Core i5-3437U CPU @ 1.90 GHz x 4 with 16 GB of memory.

2) Full case: The previous simple case is useful to understand the problem but not completely realistic. Indeed, the robot can meet more than one obstacle at a time, and each obstacle has a spatial extension. For example it could be in the zones $x = 2$ and $x = 3$ at the same time. This is impossible to model with a single coordinate pair (x, y).

We will therefore think in terms of zones occupied or not. Five zones are defined as in Fig. 7. Their boudaries are based on the same thresholds as in the previous (x, y) model, with possibly some grouping of the coordinate areas. For example, the warning areas at the left and in front of the robot are grouped into the same zone: they have similar observation characteristics (the presence of obstacles is observed but not their type) and the previous analysis suggests that they call for similar safety rules.

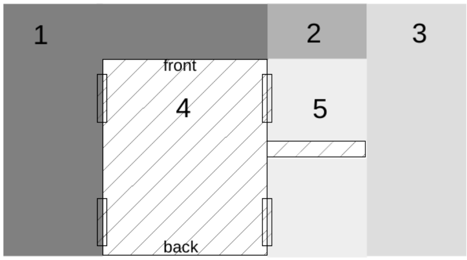

Fig. 7. Obstacle zones to avoid collision with an obstacle

Zone z_1: Front and left side of the robot	Empty	$z_1 = 0$
	Occupied by at least one obstacle (type unknown)	$z_1 = 1$
Zone z_2: Front right side of the platform	Empty	$z_2 = 0$
	Occupied by at least one high obstacle	$z_2 = 1$
	Occupied by low obstacles only	$z_2 = 2$
Zone z_3: Right side of the robot	Empty or occupied by low obstacles only	$z_3 = 0$
	Occupied by at least one high obstacle	$z_3 = 1$
Zone z_4: The platform itself	Empty	$z_4 = 0$
	Occupied by an obstacle (physical contact)	$z_4 = 1$
Zone z_5: Right side of the platform	Empty	$z_5 = 0$
	Occupied by at least one high obstacle	$z_5 = 1$
	Occupied by low obstacles only	$z_5 = 2$

TABLE V
PARTITIONING OF ZONE VARIABLES FOR THE FULL CASE MODEL

Table V shows the partitioning of zone variables into occupation classes. Value 0 encodes an empty zone. The meaning of other values depends on whether or not the type of obstacles is observable in corresponding zone. For example, variable $z1$ is two-valued with 1 indicating the occupation by obstacles of unobserved type. Variable $z2$ is three-valued in order to distinguish occupation by at least one high obstacle ($= 1$), and occupation by low obstacles only ($= 2$). The partitioning of $z3$ is specific: while the type of obstacles is observed in this zone, the variable is two-valued. The case of occupation by low obstacles only is safe for this zone, so we decided to put it in the same class as emptiness. In practice, note that the occupation cases are determined by combining the results of several sensors. Assume that the implementation is like in Fig. 5: occupation by low obstacles only would be determined by the fact that the 2D lidar on the platform's corner sees something in the zone, while the lidar on the deported extension (placed at a higher height) sees nothing.

This encoding focuses on the occupation of zones, and misses information on their spatial adjacency. In the simple (x, y) model, adjacency was easily captured by the continuity constraints on x and y. The notion of warning state then naturally emerged, since the obstacle is always seen with coordinates in a close area before getting too close. In the new model, we have to introduce ad hoc constraints to represent this. For example, a constraint expresses the fact that $z4$ cannot be occupied if, at the previous step, it was empty and all its neighboring zones were empty as well. A similar constraint is given for $z5$. The stationary assumption for low obstacles is also quite difficult to model. In particular, we have to capture the fact that, if the robot is stopped, low obstacles cannot appear and disappear in its neighborhood. It is done by introducing specific auxiliary variables and constraints. For example, we detect a situation in which the robot stops with low obstacles only in $z5$, and constrain this zone to take only non-empty values as long as the robot remains standstill.

The velocity variable remains the same as in the previous (x, y) model. We define the catastrophic state as follows:
`cata: v=1 & (z4=1 | z5=1);`

The SMOF tool identifies 94 warning states before invariant violation (vs. 16 previously). They account for the combinations of all occupation cases of the zones.

The two available interventions are the same as previously.

The search space of candidate strategies is very large, and the rule synthesis tool does not succeed in computing the set of solutions in a realistic amount of time. The tool also has a fast exploration mode at the expense of potentially skipping solutions. In the case of this model, the fast exploration manages to proceed the model in 4 minutes but finds no solution. We then manually encoded the strategy suggested by the analysis of simple case model:

- Brake when a high obstacle gets too close to the robot i.e. enters one of the zones:
  `trigger_full_stop: z1=1 | z2=1 | z3=1 | z4=1 | z5=1;`
- Restrict curve when a low obstacle is in front, behind or below the sensor support i.e. in the zone 2 or 5:
  `trigger_restrict_right_curve: z2=2 | z5=2;`

NuSMV confirms that this strategy is safe and universally permissive. Note that there are now states for which both interventions are active at the same time, e.g., when both $z_1 = 1$ and $z_2 = 2$. This is indeed necessary because the full stop does not prevent from collision with a low obstacle in z_2, which can be closer than the braking distance.

D. Consistency analysis

Once satisfying strategies are found for all invariants, we have to verify their consistency. Indeed, each invariant has been modeled separately but the observations and interventions may not be independent. We aggregate the models from the different invariants and encode the links between them in a global model. The gluing logic takes care that if the monitoring of collisions triggers the full stop, then the full stop is triggered as well in other invariant models with an effect that depends on the local preconditions. Variables that appear in several models are connected by manually adding constraints. For example, velocity cannot be zero in the monitoring of collisions and at the same time be above the warning threshold for the Invariant 5 identified in Section III. Note that forgetting constraints would keep spurious states into the global model. This is an issue for the verification of global permissiveness, which could be too optimistic.

Two main aspects have to be verified on this global model.

The first one is the absence of conflicting interventions, which would be triggered concurrently with an inconsistent effect (e.g., both braking and accelerating). In our opinion, none of the interventions we introduced would be incompatible with the other ones, and indeed the formal verification finds no inconsistency. There are cases for which several interventions are active but this is not a problem. For example, NuSMV allowed us to find additional cases where both braking and right curve restriction are needed, this time from the concurrent decision of *different* monitors.

The second aspect is the overall permissiveness. Depending on the intervention retained for prohibited areas (the full stop or the selective restriction of movement), the overall model is found universally or simply permissive, hence revealing no other issue than the ones we already identified.

V. RELATED WORK

Safety concerns have an increasing importance in robotics research. For instance, several recent European projects consider safety as a main challenge, like [9]–[11]. National projects can also be mentioned in the UK [12] , in Germany [13] or in the US [14]. From an industrial perspective, a few specific safety-related standards have been released: ISO 10218:2011 [15], [16] for robots in industrial environments and ISO 13482:2014 [17] for personal robots. A standard dedicated to collaborative robots is under development, currently named ISO/TS 15066 Robots and robotic devices – Safety requirements for industrial robots – Collaborative operation [18].

The safety monitoring approach studied in this paper is an instance of a well-established principle for safety-critical systems: clearly identify the safety functions and separate them from the main application ones (see, e.g., [19]). It makes the certification of systems easier, since only those safety functions (and not the whole system) are assigned a high level of integrity. Safety monitors – i.e., external and independent mechanisms that force the system into a safe state, should some hazardous behavior be detected – have been used for robotic systems under many different names: *safety manager* [20], *autonomous safety system* [21], *checker* [22], *guardian agent* [23], or *emergency layer* [24]. In most of the cases, the specification of the safety rules is ad hoc. In contrast, our approach provides a complete safety rule identification process, starting from a hazard analysis and using formal verification techniques to synthesize the rules. In [25], the authors also use HAZOP to identify hazards and then (intuitively) determine a set of If-then-else safety rules.

Safety monitoring is related to runtime verification and property enforcement. Runtime verification [26], [27] checks for properties (e.g., in temporal logic) by typically adding code into the controller software. Runtime enforcement [28] extends runtime verification with the ability to modify the execution of the controller, in order to ensure the property. These techniques consider a richer set of property classes than safety ones, and, most importantly, can be tightly coupled to the system. It makes the underlying mechanisms quite different from the external safety monitors having to rely on limited observation and intervention means.

VI. CONCLUSION

We presented in this paper a method to formally synthesize those safety rules with dedicated tools, successfully applied to an industrial case study: the mobile robot 4mob from the French company Sterela, developed for airfield lighting monitoring. The most interesting characteristic of this case study is the need to accommodate movements in close vicinity of obstacles (the lights to control).

A hazard analysis performed with HAZOP-UML led to the identification of 10 hazards. Some hazards were treated with a modification of the specification (e.g. removing the automated runway evacuation) and others with operating procedures. But 5 hazards were selected to be covered by the safety monitor and formalized by invariants. The formalization phase forces an explicit identification of implementation requirements. Indeed, we identified a set of observations and interventions to be made available to the monitor. For example, the analysis of collision highlighted the need for being able to distinguish high and low obstacles on specific areas around the robot. A technical solution currently discussed is to put lidar sensors on the deported extension, in addition to the ones originally planed on the platform corners. As regard interventions, we identified both classical interventions, like the full stop, and more specific ones to intercept and potentially modify the commands sent to the wheels. We also identified a set of thresholds, defining warning regions in which the interventions are applied. The exact calculation of these thresholds will have to be done by engineers based on the worst cases for both the dynamics of the system (e.g. maximal braking distance) and the execution time of the data processing part.

The rule synthesis algorithm, in its current version, shows some limitation as the largest model could not be successfully explored. The strategy synthesized in the simpler version of the model had to be manually introduced and was verified using NuSMV. Our previous applications of SMOF to industrial systems did not encounter this problem. They involved no invariants of the complexity of the 4mob collision avoidance one.

As part of a broader programme of work for improving the SMOF framework, we are working on improving the efficiency of the SMOF algorithm for large state spaces. The tool set will also be extended to better support the consistency analysis, using for example modeling conventions to ease the gathering of the synthesized strategies. We also intend to extend the method to a multi-level approach: a cascade of monitors to maintain an invariant, triggering interventions at several levels of the system architecture.

VII. ACKNOWLEDGEMENTS

This work has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 644400 (CPSE Labs project).

REFERENCES

[1] M. Machin, F. Dufossé, J.-P. Blanquart, J. Guiochet, D. Powell, and H. Waeselynck, "Specifying safety monitors for autonomous systems," in *SAFECOMP*. LNCS, 2014.

[2] J. Guiochet, "Hazard analysis of humanrobot interactions with HAZOP-UML," *Safety Science*, vol. 84, pp. pp. 225–237, 2016.

[3] M. Machin, F. Dufossé, J. Guiochet, D. Powell, M. Roy, and H. Waeselynck, "Model-checking and game theory for synthesis of safety rules," in *16th IEEE International Symposium on High Assurance Systems Engineering, HASE 2015, Daytona Beach, FL, USA, January 8-10, 2015*, 2015, pp. 36–43.

[4] M. Machin, G. Jérémie, W. Hélène, J.-P. Blanquart, M. Roy, and L. Masson, "Smof - a safety monitoring framework for autonomous systems," *IEEE Transactions On System, Man and Cybernetics: Systems*, Accepted for publication.

[5] HAZOP-UML website, LAAS-CNRS, https://www.laas.fr/projects/HAZOPUML, 2015, accessed July 2015.

[6] SMOF, "Safety Monitoring Framework," LAAS-CNRS Project, https://www.laas.fr/projects/smof, 2015, accessed 2015-07-01.

[7] "Tuleap home page," https://www.tuleap.org/.

[8] "NuSMV home page," http://nusmv.fbk.eu/.

[9] SAPHARI, "Safe and Autonomous Physical Human-Aware Robot Interaction," Project supported by the European Commission under the 7th Framework Programme, www.saphari.eu, accessed 2015-05-17, 2011-2015.

[10] SAFROS, "Patient Safety in Robotic Surgery," Project supported by the European Commission under the 7th Framework Programme, www.safros.eu/safros/, 2009-2013, accessed: 2015-04-30.

[11] ROBOT-PARTNER, "Seamless Human-Robot Cooperation for Intelligent, Flexible and Safe Operations in the Assembly Factories of the Future," Project supported by the European Commission under the 7th Framework Programme, www.robo-partner.eu/, 2013-2016, accessed: 2015-04-30.

[12] ROBOSAFE, "Trustworthy Robotic Assistants," EPSRC-funded project, UK, www.robosafe.org/, 2013, accessed: 2015-07-30.

[13] SIMERO, "Safety strategies for human-robot cooperation," Partially funded by the German Research Foundation, www.ai3.uni-bayreuth.de/projects/simero/, 2003, accessed: 2015-07-30.

[14] NREC, "National Robotic Engineering Center, Carnegie Mellon University," www.nrec.ri.cmu.edu/capabilities/safety_ops/, 2015, accessed: 2015-07-30.

[15] ISO10218-1, "Robots and robotic devices – safety requirements for industrial robots – part 1: Robots," International Organization for Standardization, 2011.

[16] ISO10218-2, "Robots and robotic devices – safety requirements for industrial robots – part 2: Robot systems and integration," International Organization for Standardization, 2011.

[17] ISO13482, "Robots and robotic devices – safety requirements for personal care robots," International Organization for Standardization, 2014.

[18] ISO/TS15066, "Robots and robotic devices – collaborative robots," International Organization for Standardization.

[19] J. Rushby, "Kernels for safety," 1989, pp. 210–220.

[20] C. Pace and D. Seward, "A safety integrated architecture for an autonomous safety excavator," in *International Symposium on Automation and Robotics in Construction*, 2000.

[21] S. Roderick, B. Roberts, E. Atkins, and D. Akin, "The ranger robotic satellite servicer and its autonomous software-based safety system," *IEEE Intelligent Systems*, vol. 19, no. 5, pp. 12–19, 2004.

[22] F. Py and F. Ingrand, "Dependable execution control for autonomous robots," in *International Conference on Intelligent Robots and Systems (IROS)*, 2004, pp. 1136–1141.

[23] J. Fox and S. Das, *Safe and sound - Artificial Intelligence in Hazardous Applications*. AAAI Press - The MIT Press, 2000.

[24] S. Haddadin, M. Suppa, S. Fuchs, T. Bodenmller, A. Albu-Schffer, and G. Hirzinger, "Towards the robotic co-worker," in *The 14th International Symposium on Robotics Research (ISRR2011)*, C. Pradalier, R. Siegwart, and G. Hirzinger, Eds. Springer Berlin Heidelberg, 2011, pp. 261–282.

[25] R. Woodman, A. F. Winfield, C. Harper, and M. Fraser, "Building safer robots: Safety driven control," *Internatioanl Journal of Robotics Research*, vol. 31, no. 13, pp. 1603–1626, 2012.

[26] M. Leucker and C. Schallhart, "A brief account of runtime verification," *Journal of Logic and Algebraic Programming*, vol. 78, no. 5, pp. 293–303, 2009.

[27] N. Delgado, A. Q. Gates, and S. Roach, "A taxonomy and catalog of runtime software-fault monitoring tools," *Transactions on Software Engineering*, vol. 30, no. 12, pp. 859–872, 2004.

[28] Y. Falcone, J.-C. Fernandez, and L. Mounier, "What can you verify and enforce at runtime?" *International Journal on Software Tools for Technology Transfer*, vol. 14, no. 3, pp. 349–382, 2012.