Brock University

Department of Computer Science

Safety aspects of autonomous robot software development

J.A. Barchanski
Technical Report # CS-04-05
February 2004

Brock University
Department of Computer Science
St. Catharines, Ontario
Canada L2S 3A1
www.cosc.brocku.ca

Safety aspects of autonomous robot software development

Jerzy A. Barchanski
Dept. of Computer Science
Brock University
St. Catharines, Canada, jbarchan@cosc.brocku.ca

Abstract: This paper is concerned with safety aspects of autonomous robot software development. Autonomous robots may operate unattended and through an unsafe operation may cause significant human, economic, or mission losses. Similar problems were encountered early on in manufacturing automation; but autonomous robots may change their behavior and operate in much less controlled environments. We concentrate in this paper on the safety of robot control software. This software allows unprecedented complexity of robotic systems, which goes beyond the ability of current engineering techniques for assuring acceptable risk. Accidents arise due to robot autonomy or adaptability or as a result of the interactions among the components of the robot control architecture. We discuss shortly safety aspects of those features. We conclude the paper with description of a robot software development process taking into account safety constraints.

Keywords: system safety, autonomy, adaptability, robot, control architecture

1. Introduction

Long time ago, the famous science fiction writer Isaac Asimov [1] has formulated Three Laws of Robotics, which should be followed to protect humans from robots and the robots themselves:

1. A robot may not injure a human being, or through inaction, allow a human being to come to harm.
2. A robot must obey the orders, given it by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence, as long as such protection does not conflict with the First or Second Laws.

These Laws effectively require that before we release robots into real-world environments, we need some credible and computationally tractable means of making them obey the laws. They are, of course, too general for direct implementation and need to be operationalized into specific properties testable on robots.

This paper is concerned with those aspects of autonomous mobile robots development which should be considered to satisfy the Laws. We view robots as situated, real-time embedded systems endowed with enough intelligence to adapt to changing environment and learn from their experience. They may operate unattended and through an unsafe operation may cause significant human, economic or mission losses. Similar problems were encountered early on in manufacturing automation; but autonomous mobile robots may change their behavior and operate in much less controlled environments.

The paper is structured as follows. We define first the principal concepts of system safety: accident, hazard, risk and safety in the context of robot hardware and software. In the following, we concentrate on the safety of robot control software. This software allows unprecedented complexity of robotic systems, which goes beyond the ability of current engineering techniques for assuring acceptable risk. Accidents arise due to robot autonomy or adaptability or as a result of the interactions among the components of the robot control architecture. We discuss safety aspects of those features. We conclude the paper with description of a robot software development process taking into account safety constraints.

2. Principal concepts of system safety

Safety refers to the ability of a system to operate without causing an accident or an unacceptable loss [9]. An **accident** is an undesired and unplanned (not necessarily unexpected) event that results in (at least) a specified level of loss. To prevent accidents, something must be known about their precursors, and these precursors must be under control of the system designer. To satisfy these requirements, system safety uses the concept of **a hazard**. There are many definition of hazards. We will define a hazard as a state or set of conditions of a system that, together with other conditions in the environment of the system may lead to an accident (or loss). It means, we define a hazard with respect to the environment of the system or component.

In most cases, accidents involve the environment within which a component or system exists. If the appropriate environmental conditions do not exists, then there is no loss and, by definition no accident. For example, the release of toxic material will cause a loss only if there are people in the vicinity. Weather conditions may also affect whether a loss occurs.
In case of physical systems, existence of a hazard depends on how the boundaries of the system have been drawn – they must include the object that is damaged plus all the conditions necessary for the loss.
This does not apply to software, since it is not a physical object, only an abstraction. Thus software by itself is not safe or unsafe, although it could theoretically become unsafe when executed on a computer. Thus we can talk only about the safety of software and its hazards in the context of a particular system design within which it is being used. Otherwise, the hazards associated with software do not exists. Due to this, the system safety engineers prefer to use the term software system safety instead of software safety.
A hazard has two important characteristics: severity and likelihood of occurrence.
Hazard severity is defined as the worst possible accident that could result from the hazard given the environment in its most unfavourable state.
The hazard likelihood of occurrence can be specified either qualitatively or quantitatively. Unfortunately, when the system is being designed and hazards are being analysed and ranked as to which should be eliminated first, the information needed to evaluate the likelihood accurately is almost never available. It means, the best what can be done is to evaluate the likelihood qualitatively.
The combination of severity and likelihood of occurrence is often called the hazard level**. Hazard level**, along with two other factors related to hazards, are used in the definition of a risk. **Risk** is the hazard level combined with (1) the likelihood of the hazard leading to an accident and (2) hazard exposure or duration. Duration is a component of a risk, since the longer the hazardous state exists, the greater the chance is that the other prerequisite conditions will occur.

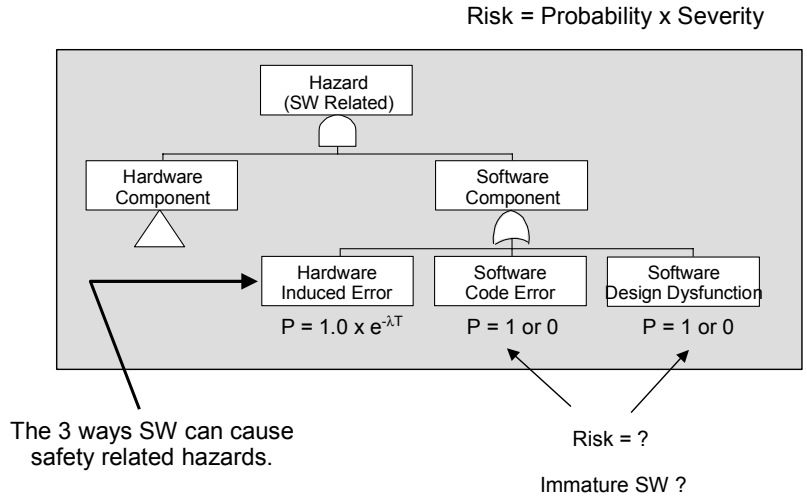

Figure 1. Generic software hazard model

The relationship between hardware and software hazards [13] is shown on Fig.1, where:
Design dysfunction means any hazard inadvertently built-into the system design due to design and integration, e.g.: design error, design interface error/oversight, design integration error/oversight, tool errors;
Code error is any hazard inadvertently built-into the system design due to a coding error, e.g.: wrong sign (+/-), endless loop, language error;
Hardware induced software error - any hazard resulting from hardware failure causing safety critical software error, e.g. memory error, memory jump, bit error, change of value of a critical variable.

As an example of risk evaluation let's take the case when a computer controls movements of a robot.
The risk is then a function of the

1. probability that the computer causes a spurious or unexpected robot movement,
2. probability that a human is in the field of movements,
3. probability that the human has no time to move or will fail to diagnose the robot failure,
4. severity of worst-case consequences.

If a computer executes a robot control software monitoring state of a robot and including some safety function, then the risk is a function of the

1. probability of a dangerous plant condition arising,
2. probability of the computer not detecting it,
3. probability of the computer not initiating its safety function,
4. probability of the safety function not preventing the hazard,
5. probability of conditions occurring that will cause the hazard to lead to an accident,
6. worst-case severity of the accident.

In almost all cases, the correct way to combine the elements of the risk function are unknown, as are the values of the parameters of the function.
Sometimes, the terms risk analysis and hazard analysis are used interchangeable, but an important difference exists. Hazard analysis involves only the identification of hazards and the assessment of hazard level, while risk analysis adds the identification and assessment of the environmental conditions together with duration. Thus hazard analysis is a subset of risk analysis.

3. Safety aspects of autonomous adaptive robots

3.1. Autonomy

The concept of autonomy plays an important role in robotics. It relates to an individual or collective ability to decide and act consistently without outside control or intervention Autonomous robots holds the promise of new operation possibilities, easier design and development and lower operating costs.
There are two main reasons driving the development of autonomous robots: one is budgetary, the other is technical [12]. On the budgetary side, autonomy will reduce the need for human attendance. Even more importantly, though, autonomy reduces the reliance on the communication link between the robots and their operators, opening a full new range of opportunities:
- in space, information takes more and more energy and time to reach its destination as distance increases – up to 20 minutes from Earth to Mars. Local autonomous control software will enable much faster reactions. This can increase productivity, enable new missions or even save the life of a spacecraft.
- even with negligible communication delays, autonomy can provide computer-speed reaction times in places where human response time would be too slow with respect to the environment, as for example in collision avoidance systems.
- autonomy also continues to work when no communication is possible at all, because of interference or physical obstacles. In planetary missions, this happens when a spacecraft passes behind a planet; in earth-bound missions, this enables deep underground or submarine explorations.
Achieving safety of autonomous robots is much more challenging than teleoperated robots for several reasons:
- First, autonomous robots control systems close the control loops and arbitrate resources on-board, making it more difficult to plug in test harnesses and write detailed test runs that drive the system through a desired behavior.
- Second, the range of situations to be tested is incomparably larger. In the open loop case, it is up to the intelligence of the experts in control to choose the appropriate response to a situation as it occurs. In the autonomous case, the program implicitly incorporates response scenarios to any combination of events that might occur. The reaction can also depend on the current configuration of the system, or even on its past history in the case of adaptive systems. All these factors multiply exponentially with the size of the system, and a test suite can only exercise a very limited portion of those cases.
- Third, as different concurrent parts of the autonomous controller interact together internally, the controller can now react in different ways to the same stimuli, for example because of differences in scheduling. The consequence is that a successful test run does not even guarantee that the system will behave correctly for the tested scenario, because the same input sequence can lead to different execution sequences. Another test run could fail due to uncontrollable changes of circumstances. This is a well-known issue in designing concurrent systems. To improve safety of autonomous systems, adjustable autonomy can be used, in which the robot is autonomous to some degree only so a human may still retain more or less control of its behavior. While verification of the safety of a fully autonomous robot designed for a critical mission requires extensive test and analysis, safety verification of a semi-autonomous robot designed for the same mission is less strict – it includes only design requirements analysis and testing.

3.2. Adaptability

Designers can furnish robots with plans to perform desired tasks, however they cannot foresee all circumstances that will be encountered by the robots. Therefore, in addition to supplying a robot with a plan, it is essential to enable it to learn and modify its plan to adapt to unforeseen circumstances.
Researchers have worked on achieving adaptive behavior through various approaches, from traditional adaptive control systems, to more recent neural networks and genetic algorithms. Many of those approaches fall under the broad and in the last decade in particular, exceedingly popular approach called "reinforcement learning" [3]. In this approach a robot receives an external feedback from the environment, interprets it as positive or negative scalar reinforcement and adapts appropriately.
The introduction of learning however, often makes the robot's behavior significantly harder to predict.
Any kind of a priori analytic verification of safety becomes problematic. A solution could reside in run-time incremental verification, where the parts of the system affected by an adaptive re-configuration are verified before the re-configuration is committed. Since the verification has to be performed on-board and during the operation of the autonomous system, this may put very tight time and space requirements on the verification process..
More realistic solution is development of methods for determining whether the behavior of a learning robot remains within the bounds of prespecified safety constraints after learning. In case of genetic algorithms e.g. it is necessary to ensure that the genetic operators will preserve robot safety [6]. If the robot uses these "safe" learning operators, it will be guaranted to preserve safety with no reverification required. This is the best one could hope for in an online situation where rapid response time is critical.
For other learning operators it is possible to develop incremental reverification algorithms that can save time over total reverification from scratch.

3.3. Control Architecture

The term Robot Control Architecture is usually used to mean a set of functional components of a robot control system and the principles of their interaction [2]. In addition to providing a structure it imposes constraints on the way a robot control problem can be solved. The components of an architecture are implemented mostly in software..
Traditionally the architectures were of a hierarchical type, highly influenced by the AI research of the time. This meant a system having an elaborate model of the world, using sensors to update this model, and to draw conclusions based on the updated model. This is often called the sense-plan-act paradigm or deliberate architecture. The architectures did not perform very well, partly because of the difficulty in modeling of the world, partly because of relying too much on inadequate sensors. They are slow, inherently sequential, require large memories, are not scalable, and are not useful in a dynamic and noisy environment.
In 1987 Rodney Brooks [5] revolutionized the field by presenting an architecture based on purely reactive behaviors with little or no knowledge of the world. The reactive architecture is inherently parallel, fast, operates on short time scales and is scalable However, the purely reactive scheme does not perform well when performing complex tasks like planning and goal reaching.
A hybrid approach [2] has since then been common among researchers, which combines different representations and time scales, combines closed-loop and open loop execution, may re-use plans and allows dynamic replanning. The disadvantage is that it is hard to design and it is not scalable.
As we said at the very beginning, system safety does not depend so much on the individual system components, but on their interaction. So, selection of a right control architecture is crucial for robot safety. The present research on robot control architectures does not follow the methodology of Safeware Engineering [9]. As a result the robots may be robust beyond a need (or not) but their level of safety is actually unknown, because it cannot be evaluated. We are analyzing at present several robot control architectures to uncover their unsafe design aspects, to select the safest and to create an architecture with a specified level of acceptable risk. .

4. Development of safe robot control software

Development of safe robot control software should proceed according to the following process:

4.1. Hazard analysis

The process should start with identification of hazards which may be encountered by mobile robots in a specific operational environment. The results of hazard analysis are critical to the process of development for safety and verification that the implemented system is safe.
Hazard analysis usually requires searching for potential sources of hazards through large system state spaces. To handle this complexity it is possible to use a State Machine Hazard Analysis [11]. While any state machine modeling language can be used, it is more efficient to use a language providing higher-level abstractions, such as Statecharts [7].

4.2. Risk analysis

The next step is to carry out risk analysis. Similarly as hazard identification, it is done at present by domain experts and is subject to their individual perception and biases. More research is necessary on qualitative risk assessment, using for example fuzzy logic.

4.3. Intent specification

Following this, it is necessary to specify desired features of the robot in a format called intent specification [10]. Most accidents related to software stem from conventional requirements specification flaws -- incorrect assumptions about the required behavior of the software and the operational environment. In almost all accidents involving computer-controlled systems, the software performed according to specification but the specified behavior was unsafe.
Intent specifications employ a type of hierarchical abstraction based on intent or purpose. They include, among others requirements, environmental assumptions and safety constraints. The specifications support safety-driven development by tightly integrating the system safety process and the information resulting from it into the system engineering process and decision–making environment. The goal is to support design of safe systems rather than simply the attempt to verify safety after-the–fact. Safety-related design decisions are linked to hazard analysis and design implementations so that assurance of safety is also enhanced as well as any analysis required when changes are proposed.

4.4. Design specifications

Intent specifications are mapped into design specifications of a selected robot control architecture. The mapping can be done in one of three ways:
1. Uncoupled: the mappings or functions from requirements to design principles are all one-to-one.
2. Loosely coupled: the mappings are one-to-many.
3. Tightly coupled: the mappings are many-to-many.

For any complex control system, a completely uncoupled designs, while allowing changes to requirements with minimal impact, is usually not practical.
A tightly-coupled system is highly interdependent. Each part is tightly linked to many other parts, so the that a failure or unplanned behaviour in one, can rapidly affect the status of others. This is definitely not desirable.
The most appropriate and realistic is to design the system in such a way as to reduce the impact of requirements changes, i.e. to eliminate the rippling effects to other requirements, by designing the mappings to be one-to-many (loosely coupled). By this, a failure of a design element can be traced to incorrect or incomplete requirement.

4.5. Simulation

For a long time simulation was not recommended for robotics, due to robot embodiment and situatedness, environment dynamics and uncertainty, noise in sensors and actuators. As it is impossible to model the robot in its environment faithfully it was often said that the world is its own best model [5] and it is best to experiment with a real robot.
With time this opinion was relaxed somehow, and it was accepted that simulation may be useful e.g. for assessing approximate performance of a robot.
For practical reasons simulation does make sense if its results can be transferred to a real robot (are transferrable). Some researchers suggested that to achieve transferrability, every effort should be made to keep simulations in close step with reality. It was suggested moreover that it is necessary to:
(a) base the simulation on carefully collected empirical data of a real physical robot;
(b) add noise to the simulated sensory readings and the actuators outcomes; and
(c) calibrate the simulation through tests on the real physical robot.
Such suggestions are not much of use during robot design, when the physical robot is not available yet.
Some researchers argue that if behavioral transferrence can only be guaranteed when a carefully constructed empirically validated simulation is used, then as robots become more complicated, so do the simulations. The level of complexity involved, they argue, would make such simulations:
- so computationally expensive that all speed advantages over real-world experimentation are lost;
- so hard to design that the time taken in development outweighs time saved by fast simulation.

Clearly the main challenge for using simulation in robotics is to invent a general theoretical and methodological framework that enables an easy and cheap construction of fast running simulators. Such a methodology was proposed for evolutionary robotics

(minimal simulation approach [8]). We have applied minimal simulation to hand-designed behavior-based robot control architecture [4]. The model of a robot controller satisfying the conditions of the minimal simulation can be transferred then to a physical robot.

4.6. Implementation and testing

The final step is the implementation and testing of the selected control architecture on physical robot in the specified operational environment.. While the previous stages of the development process can be accomplished with modeling and simulation, this stage requires access to a physical robot.
The purpose of testing is to ensure that all safety requirements are implemented and effective, to ensure safety of code changes and to determine code maturity. It allows validation of all the preceding steps and verification of the level of safety.
These goals can be achieved via design testing (module/units), integration testing (subsystems), and system testing.

5. Conclusion

Knowledge of how to build safe autonomous mobile robots is a prerequisite for their wide acceptance. In this paper we have described the relationship between robot autonomy, adaptability and its safety. We have reviewed as well the most popular at present robot control architectures and indicated the differences between them which may reflect on their safety.
Moreover, we have outlined a methodology for building safe autonomous mobile robots – including hazard analysis, risk analysis, intent specification, design, simulation, implementation and safety verification

References:

[1] Isaac Asimov, I Robot, 1950.
[2] Ronald Arkin, Behavior-Based Robotics, MIT Press 1998,
[3] J. A. Barchanski, Matthew Pratola, Adaptive Tracking and Capture by Behavior-Based Mobile Robotic Teams, 3rd International Symposium on Robotic and Automation, Sept. 2002, Toluca, Mexico.
[4]. J.A. Barchanski, Simulation Environment for Behavior-Based Robotic Teams, ESM2001, European Simulation Multiconference, Prague, Czech Republik, 5-9 June 2001
[5] Brooks, R. A Robust Layered Control System for a Mobile Robot, IEEE Journal of Robotics and Automation, Vol. RA-2, No 1, 1986.
[6] Diana F. Gordon-Spears, Assuring the behavior of adaptive agents, Book chapter in *Agent Technology from a Formal Perspective.* Kluwer, 2003, prepublication version,
[7] David Harel, Statecharts: A Visual formalism for complex systems, Science of Computer Programming, 8:231-274, 1987.
[8] Jacobi, N., Minimal Simulation for Evolutionary Robotics, Ph. D. thesis, May 1998.
[9] Nancy G. Leveson, Safeware: System Safety and Computers, Addison Wesley, 1995.
[10] Nancy G. Leveson, Intent Specification: An Approach to Building Human-Centered Specifications, IEEE Trans. on Software Engineering, January 2000.
[11] Nancy G. Leveson and Janice L. Stolzy, Safety Analysis Using Petri Nets, IEEE Transaction on Software Engineering, March 1987,
[12] Charles Pecheur, Verification and Validation of Autonomy Software at NASA,.
[13] Software Safety Summit, 23 May 2003.