2

# Development of Safe and Secure Control Software for Autonomous Mobile Robots

Jerzy A. Barchanski
*Brock University*
*Canada*

## 1. Introduction

It is often said that autonomous robots should be *trustworthy* or *dependable*, meaning by this safe, secure, reliable, etc This terms are too general to be useful, so we prefer to limit the scope of this chapter to two of their inter-related components - safety and security. They must be built into a system from the start; it is difficult, if not impossible, to add them in an adequate and cost-effective manner later on.

We view autonomous 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. The focus of this chapter is on safety and security 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.

Most of the publications on safety has a form of recommendations on providing safe environment for robot operators, like the Occupational Safety and Health Administration regulations or the more recent NASA recommendations for space robots. This approach is effective when accidents are primarily caused by hardware components failures.

As software becomes more and more important in robot control, we have to consider ways to prevent accidents caused by software.

Robot control software consists of many interacting components. Accidents arise in the interactions among the components rather than the failure of individual components.

The need for safety is obvious, but how to ensure it is less obvious. 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 mobile robots may change their behaviour and operate in much less controlled environments.

We will review at first the principal concepts of system safety like risk and hazard and some traditional approaches to dealing with them. We consider security as a subset of safety and we will present our point of view on this issue.

The present trend to make the robots more autonomous requires new approaches to deal with much more complex problems of their safety. After review of several robot control architectures from the viewpoint of their safety we present an approach based on systems theory. While the theory was developed long time ago it turns out very useful to ensure

safety of complex software systems. We apply it in so called intention specification, which allows a designer to explain why a specific decision was made and to introduce 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 (Leveson 1995). 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 definitions 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). We define therefore a hazard with respect to the environment of the robot.

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 exist. 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 hazards, are used in the definition of 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 greatest the chance is that the other prerequisite conditions will occur.

The relationship between hardware and software hazards 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.

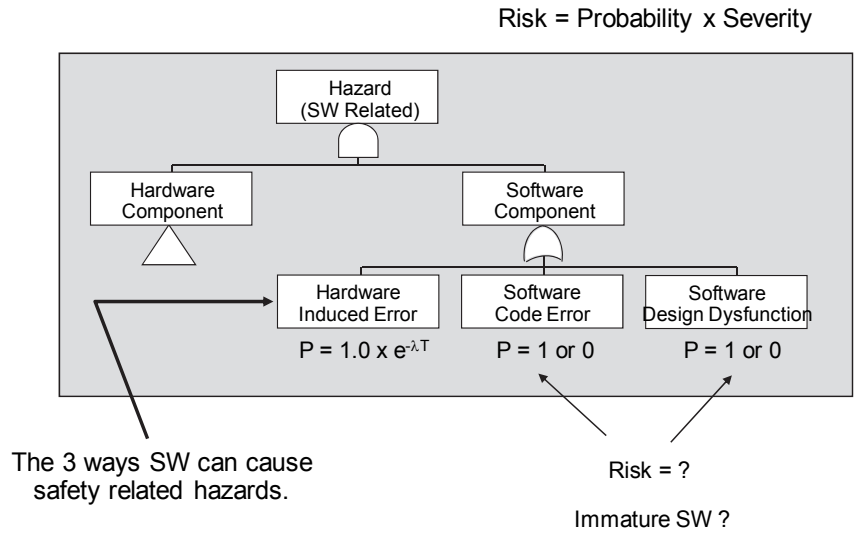



Fig. 1. Generic software hazard model

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

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

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

- probability of a dangerous condition arising,
- probability of the computer not detecting it,
- probability of the computer not initiating its safety function,
- probability of the safety function not preventing the hazard,
- probability of conditions occurring that will cause the hazard to lead to an accident,
- 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.
Traditional hazard analyses consist of identifying events that could lead the system to a hazardous state. These events are usually organized into causal chains or trees. Popular event-based hazard analysis techniques include Fault Tree Analysis (FTA) and Failure Modes and Effects Criticality Analysis (FMECA). Because of their reliance on discrete failure events, neither of these techniques adequately handles software or system accidents that result from dysfunctional interactions between system components.
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 (Leveson 1987). While any state machine modeling language can be used, it is more efficient to use a language providing higher-level abstractions, such as Statecharts (Harel 1987).

The tasks of the software development process that relates to software hazard analysis include:
- Tracing identified system hazards to the software-hardware interface and then into requirements and constraints on software behavior.
- Showing the consistency of the software safety constraints with the software requirements specification and demonstrating the completeness of the software requirements specification.

In addition, because software can do more than what is specified in the requirements (the problem of unintended functions), the code itself must be analyzed to ensure that it cannot exhibit hazardous behavior.

## 3. Security as a subset of safety

Safety and security are closely related, and their similarity can be used to the advantage of both (Barchanski J.A., 2004),. Both deal with threats or risks – one with threats to life or property, and the other with threats to privacy or security. Both often involve negative requirements or constraints that may conflict with some important system goals. Both involve protection against losses, although the type of losses may be different. Both involve global system properties that are difficult to deal with outside of the system context. Both involve requirements that are considered of supreme importance in deciding whether the system can and should be used – that is particularly high level of assurance may be needed, and testing alone is insufficient to establish those levels. In fact, a higher level of assurance that a system is safe or secure may be needed than that system performs its intended functions.

While we will consider mostly the issues of illegitimate access and usage of a robot there is another aspect less often encountered – denial of access. It is a threat in which the communication channel is made unusable by an attacker who transmits noise on purpose (jamming). While it is not possible to survive jamming when the attacker is all powerful and can make the entire spectrum unusable over the spatio-temporal range of interest, it may be possible to drive up the cost of jamming. Two widely used techniques for this are frequency hopping and direct sequence spread. In both techniques the receiver must know the pseudorandom sequence of frequencies used by the transmitter. An attacker wishing to jam the channel must either discover the sequence or jam a sufficient number of frequencies, therefore employing much more power (and money) than the transmitter.

It should be clear that this protection is conditional on transmitter and receiver having previously established some shared key (e.g. by imprinting as described in the following paragraph).

There are also important differences between safety and security. Security focuses on malicious actions, whereas safety is also concerned with well intended actions. In addition, the primary emphasis in security traditionally has been preventing unauthorized access to classified information, as opposed to preventing more general malicious activities. Note, however, that if an accident or loss event may be caused by illegitimate or malicious access or usage of a system, then security becomes a subset of safety.

## 4. Safety aspects of autonomous robots

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 hold the promise of new operation possibilities, easier design and development, and lower operating costs.

Achieving safety of autonomous robots is much more challenging than teleoperated robots.

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.

Mobile robots control software can implement different robot control architectures.

The oldest architectures were of a hierarchical type, highly influenced by the AI research of its 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 deliberative architecture. The hierarchical architectures did not perform well partly because of the difficulty in modeling of the world, partly because of relying too much on inadequate sensors.

In 1987 Rodney Brooks revolutionized the field by presenting an architecture based on purely reactive behaviours with little or no knowledge of the world. The reactive architecture is inherently parallel, fast, operates on short time scales and is scalable. There are two general kinds of reactive architectures – subsumption and potential field (Arkin 1999). Reactive architectures eliminate planning and any functions that involve remembering or reasoning about the global state of the robot relative to its environment. That means that a robot cannot plan optimal trajectories, make maps, monitor its own performance or even select the best behaviours to use to accomplish a task (general planning).

The solutions to the drawbacks of reactive architectures appeared in hybrid architectures combining the reactive architectures with modified deliberative architectures. They combine different representations and time scales, combine closed-loop and open loop execution, may re-use plans and allow dynamic replanning [Murphy 2000].

There is a number of other robot architectures designed independently of the above classes (e.g., Alami et al, 1998). Especially interesting are architectures including learning or adaptive components. The ability to adapt is key to survival in dynamic environments. When robots are adaptive, the following questions need to be addressed:

1. Is learning applied to one or more robots?
2. If multiple robots are adaptive, are they competing or cooperating?
3. What element(s) of the robot(s) get adapted?
4. What techniques are used to adapt?

Learning may alter some components of robot strategy, for example, it may change the choice of actions to take in response to a stimulus. Learning may be motivated by observation, it could be initiated in response to success/failure, or be prompted by a general need for performance improvement. The motivation influences the choice of learning technique. Nearly every learning technique has been used for robots. The two most prevalent techniques are reinforcement learning (RL) and evolutionary algorithms (EA).

A very popular form of RL is Q-learning, where robots update their probabilities of taking actions in a given situation based on penalty/reward.

Introduction of learning makes behavior of robots significantly harder to predict. It is necessary therefore to develop efficient methods for determining whether the behavior of learning robots remains within the bounds of prespecified constraints (properties) after learning. This includes verifying that properties are preserved for single robots as well as verifying that properties are preserved for multirobot systems. We want the robots to be not only adaptive, but as well predictable and timely. Predictability can be achieved by formal

verification while timeliness by streamlining reverification, using the knowledge of what learning was done. Fast reverification after learning must include proofs that certain useful learning operators (in case of EA) are a priori guaranteed to be "safe" with respect to some important classes of properties, i.e. if the property holds for the robot prior to learning, then it is guaranteed to still hold after learning ( Gordon-Spears, 2001). If a robot uses these "safe" learning operators, it will be guaranteed to preserve the properties with no reverification required.

## 5. Hazard analysis of an exemplary case

As an example we will consider a mobile robot acting in a world where the three Laws of Robotics (Asimov 1950) should be satisfied. We will be concerned with the hazards resulting from the first two Laws, which are relevant for human-robot interactions :
First Law: "A robot may not injure a human being, or through inaction, may not allow a human being to come to harm".
From the first half of the Law: "A robot may not injure a human being" – the hazard is a harmful physical contact with a human.
It can be mitigated by a safety constraint inherent in the robot behavior **avoid _obstacles.**
The second part of the law :"A robot through inaction, may not allow a human being to come to harm" envisioned by Asimov as a kind of action requiring sacrifice of robot existence to protect its master, is left for future generations of roboticists. At present it may be implemented at most aposteriori by search and rescue robots.
The second law: "A robot must obey the orders, given to it by human beings except where such orders would conflict with the First Law."
This law has to do with tasks commanded by a human, like: **move-to-goal**, **grasp**, **collect-pucks**, etc. The robot must be able to communicate with a human operator to execute his orders. The examples of hazards violating this law are communication security hazards – e.g. receiving from an illegitimate operator false commands or requests which may cause wrong actions. To eliminate such hazards it must be possible to authenticate the human operator.
Most conventional authentication protocols (e.g. Kerberos) require usage of an online server. This is out of the question here. Another widely used authentication protocol which seems to be more suitable is one based on public key cryptography. It was in fact proposed for authentication of cooperating autonomous digger and dumper truck [Chandler et al 2001]. However the problem of online access to the server appears here as well. What we need is to be able to create a *secure transient association.* As well as being *secure*, the association must also be *transient*. When an operator changes, the robot has to obey the new operator. A solution for this dilemma is to use a metaphor of a duckling emerging from its egg – it will recognize as its mother the first moving object it sees that makes a sound, regardless of what it looks like: this phenomenon is called **imprinting.** Similarly, our robot will recognize as its owner the first entity that gives it a secret key. As soon as this imprinting key is received, the robot is no longer a newborn and will stay faithful to its owner for the rest of its life. The association is secure.
To make the association transient, the robot must be designed so it can die (loose its memory) and resurrect (be imprinted again). This security policy model, called "Resurrecting Duckling" (Stajano 2002) can be used not only in the communications between a human and a robot but between two robots as well, enabling robot-to-robot secure interaction.
We will focus in the following on the hazard involving collision with an obstacle, whether it is a human or not. To deal with this hazard, a robot should be able first to detect the

obstacle, then recognize it and finally execute the avoidance manoeuvre. Obstacle recognition is useful only if the robot is supposed to cooperate with a human or another robot. Recognition of a human is quite a difficult task. It is necessary to use for this some special sensors (e.g. a suite of vision and thermal sensors). Recognition of another robot may be easier as it is possible to give the robot a special appearance (e.g. color). To reduce the risk of collision, robots must have some kind of a behavior to avoid obstacles. This behavior should be active all the time, concurrently with any other behavior active at the moment. We will discuss in the following how efficient are the obstacle avoidance behaviors in different robot control architectures.

## 6. Safety of different robot control architectures

### 6.1 Reactive architectures

As we have noticed above, the hierarchical architecture does not provide adequate functionality for obstacle avoidance. Much better equipped for this purpose are reactive architectures. The main reason for this is their direct connection of sensors to actuators. Even though the architecture does not have a memory-based world model, but as Brooks said "the world itself is its best model." Continuous sensing guarantees that it is always current, though not always correct, due to the sensors errors or failures. One of the characteristics of reactive architectures is their ability for graceful degradation of emergent behavior (Jones 2004). Let us consider a mobile robot moving to a goal and equipped with a number of different sensors. In the simplest case it will not discriminate between different kinds of obstacles. Failure of a sensor to detect an obstacle is called false negative. A false positive occurs when a sensor reports a condition that does not exists. From the viewpoint of safety a false negative implies a hazard to be dealt with. One way to mitigate this hazard is to use a set of different sensors invoking suitable behaviors. For example a robot may use for long distance journey a **sonar-avoid** behavior. While still some distance off, the robot will turn away from sonar detected obstacles. But sonar sensors are easily fooled. Smooth surfaces struck at shallow angles reflect sonar beams forward, not back toward the robot – thus no echo is ever detected, causing the sonar system to report a false negative. When this happens, the robot believes that the path is clear, even if it is not. In this case, the **sonar-avoid** behavior fails to trigger. But as the robot approaches the acoustically specular object, typically the infrared (IR) detectors will sense an obstacle and drive the robot away. But perhaps along with being too smooth and set at too oblique an angle, the obstacle's surface is also too dark to reflect IR radiation reliably back to the IR obstacle detector. Thus the IR detector may also fail to sense the object, generating a false negative of its own.

With no signal from the IR detector, the **IR-avoid** behavior does not trigger and the robot collides with the obstacle. The robot bumper now compresses and triggers the **Bumper-escape** behavior. Having failed to avoid the obstacle, the robot must now back up and try to turn away. Typically, the bump sensors can detect, at least crudely, where the collision has occurred, say, on the right, or the left of the robot. This knowledge can give the robot an indication of how to respond.

The performance of the robot may suffer, but the robot can still continue its mission. But suppose there is yet another difficulty, say, that the bumper is not a full-coverage bumper or that it has a dead spot at some point and it is exactly that point that the troublesome smooth, dark oblique object contacts the bumper. Now the bumper fails to report the collision and accordingly, the **Bumper-escape** behavior never triggers. We are left with our robot pressing against the object with all its might. Fortunately, an over-current sensor is available to detect this condition. When the drive motors are asked to work too hard, as can happen, when

they force the robot against an object, motor current goes up. Too high current for too long a time is sensed and is used to trigger a **Stall-escape** behavior. Although there is no reliable way to tell where the blockage is located with respect to the robot, **Stall-escape** can at least command the robot to retreat and spin. So, the robot still can move towards its goal. We have assumed that all the problems are caused by a difficult-to-detect obstacle. But the same behavior would be produced by some inoperative sensors.
Note that the emergent desirable behavior does not require writing a special code that would explicitly instruct the robot to determine if a sensor is working properly. It is just the feature of behavior-based architecture.

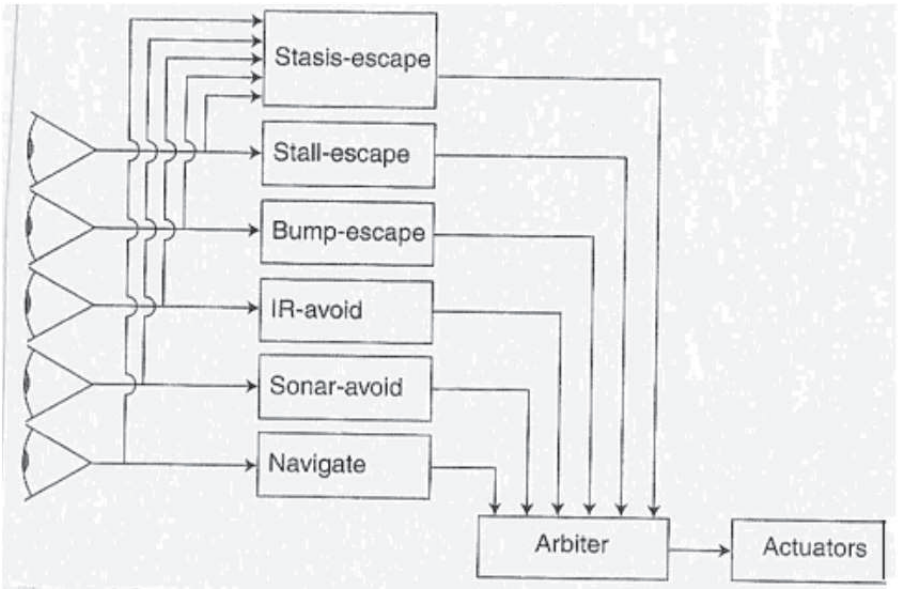



Fig. 2. Reactive architecture with graceful degradation.

The above examples show a crucial difference between the behavior-based approach of reactive architectures and the traditional deliberative approach. In reactive architectures we do not employ a single expensive sensor from which we must demand unattainable levels of precision and reliability. Rather we achieve superior results using a combination of relatively unreliable systems that work together to deliver safe behavior.
In the original subsumption architecture the speed was decided by the behavior subsuming all the other behaviors. Quite often the speed was fixed – the same for all the winning behaviors.
In case of reactive architecture using potential fields for coordination of behaviors (Arkin 1999) the emergent speed is the magnitude of the vector summed from all the active behaviors. It allows therefore to avoid obstacles while moving towards a goal – providing better overall behavior than subsumption architecture.

### 6.2 Hybrid architectures

The reactive architectures do not have an ability to monitor performance of the robot or to use e.g. an optimal speed and path to a destination, while avoiding obstacle. This can be

mitigated by appropriately designed deliberative layer of a hybrid architecture. For example, in managerial architectures (Murphy 2000), the deliberative layer may include a Sensing Manager monitoring performance of the robot. If a behavior fails or a perceptual schema detects that sensor values are not consistent or reasonable, the Sensing Manager is alerted. It can then identify alternative perceptual schemas, or even behaviors, to replace the problematic behaviour immediately. Imagine a mobile robot in a convoy of robots hauling food to refugees. If the robot had a glitch in a sensor, it should not suddenly stop and think about a problem. Instead it should immediately switch to a back-up plan or even to smoothly slow down while it identifies a back-up plan. Otherwise, the whole convoy would stop, there might be wrecks, etc. The sensing manager working in the background mode, can attempt to diagnose the cause of the problem and correct it.

In some managerial architectures (e.g. SFX (Murphy 1996) speed control is considered a separate behavior. The safe speed of a robot depends on many influences. If the robot cannot turn in place, it will need to be operating at a slow speed to make the turn without overshooting. Likewise, it may go slower as it goes up or down hills. These influences are derived from sensors, and the action is a template (the robot always slows down on hills), so speed control is a legitimate behavior. But the other behaviors should have some influence on the speed as well. So, these other strategic behaviors contribute a strategic speed to the speed control behaviour. If the strategic speed is less than the safe speed computed from the tactical influences, then the output speed is the strategic speed. But if the tactical safe speed is lower, the output speed is the tactical speed. Tactical behaviors serve as filters on strategic commands to ensure that the robot acts in a safe manner in as close accordance with the strategic intent as possible.

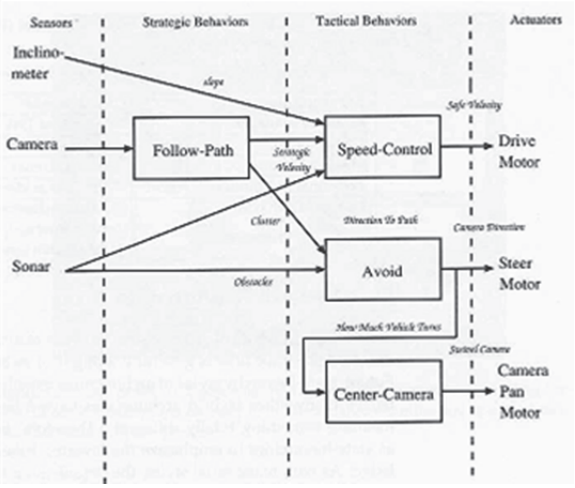



Fig. 3. Strategic and tactical behaviors for path following in SFX architecture.

An example of the Model-oriented architectures is the Task Control Architecture (TCA) (Simmons 1999). TCA has a flavor of an operating system. There are no behaviors per se, however many of low level tasks resemble behaviors. The reactive layer in this architecture called Obstacle Avoidance Layer takes the desired heading and adapts it to the obstacles

extracted from the evidence grid virtual sensor. The layer uses a curvature-velocity method (CVM) to factor in not only obstacles but how to respond with a smooth trajectory for the robot's current velocity. Because this architecture does not have direct connection of sensors to actuators, it may have the same problems as the hierarchical architectures of the past – delayed response to events.

## 7. System-theoretic approach

To make a progress in dealing with safety in complex systems, we need new models and conceptions of how accidents occur, that more accurately and completely reflect the types of accidents we are experiencing today.
We use for safety analysis of robot control architectures a system-theoretical approach (Barchanski, J.A. 2006), which allows more complex relationships between events to be considered and also provides a way to look more deeply at why the events occurred. Whereas industrial safety models focus on unsafe acts or conditions and reliability engineering emphasizes failure events, a systems approach to safety takes a broader view by focusing on what was wrong with the system's design or operations that allowed the accident to take place.
System theory dates from the thirties and forties, and was a response to limitations to the classic analysis techniques in coping with the increasingly complex systems being built. Norbert Wiener applied this approach to control and communications engineering while Ludwig von Bertalanffy developed similar ideas for biology. It was Bertalanffy who suggested that the emerging ideas in various fields could be combined into a general theory of systems (Bertalanffy 1969) .
The systems approach focuses on systems taken as a whole, not on the parts examined separately. It assumes that some properties of systems can only be treated adequately in their entirety, taking into account all facets relating the social to technical aspects. These system properties derive from the relationships between the parts of systems: how the parts interact and fit together. Thus the system approach concentrates on the analysis and design of the system as a whole as distinct from the components or the parts. While components may be constructed in a modular fashion, the original analysis and decomposition must be performed top down.
The foundations of system theory rests on two pairs of ideas:
(1) emergence and hierarchy and (2) communication and control (Checkland 1981).

### 7.1 Emergence and hierarchy

A general model of complex systems can be expressed in terms of hierarchy of levels of organization, each more complex than the one below, where a level is characterized by having emergent properties.
Emergent properties do not exist at lower levels; they are meaningless in the language appropriate to those levels. Thus, the operation of the processes at the higher levels of the hierarchy results in a higher levels of complexity – that has emergent properties.
Safety is an emergent property of systems. Determining whether a plant is acceptable safe is not possible by examining a single valve in the plant.
In fact, statements about the "safety of the valve" without information about the context in which the valve is used, are meaningless. In a system-theoretic view of safety the emergent safety properties are controlled or enforced by a set of safety constraints related to the behavior of the system components.

A second basic part of system theory, the hierarchy theory, deals with the fundamental differences between levels of a hierarchy. Its ultimate aim is to explain the relationship between different levels, what generates the levels, what separates them, and what links them. Emergent properties associated with a set of components at one level in a hierarchy are related to constraints upon the degree of freedom of those components.
In a systems-theoretic view of safety the emergent safety properties are controlled or enforced by a set of safety constraints related to the behavior of the system components.
Safety constraints specify those relationships among system variables or components that constitute the non-hazardous or safe system states. Accidents result from interactions among system components that violate these constraints – in other words, from a lack of appropriate constraints on system behavior.

### 7.2 Communication and control

Regulatory or control action is the imposition of constraints upon the activity at one level of a hierarchy, which define the "laws of behavior" at that level yielding activity meaningful at a higher level.
Hierarchies are characterized by control processes operating at the interfaces between levels. Any description of a control process entails an upper level imposing constraints upon the lower. The upper level is a source of an alternative (simpler) description of the lower level in terms of specific functions that are emergent as a result of the imposition of constraints.
Control in open systems (those that have inputs and outputs from their environment) implies the need for communication. A system is not treated as a static design, but as a dynamic process that is continually adapting to achieve its ends and to react to changes in itself and its environment. To be safe, the original design must not only enforce appropriate constraints on behavior to ensure safe operation (the enforcement of the safety constraints), but it must continue to operate safely as changes and adaptations occur over time.

## 8. Intent specification

Conventional software engineering uses top-down and bottom-up hierarchies of two kinds:
- part-whole abstractions (where each level represents aggregation of components of the lower level),
- information-hiding abstraction (where each level is a refinement of the information at a higher level).

Higher level specifies the "what", lower level specifies the "how". There is no place to specify the "why" (intent, goal, purpose) of a level.
Systems-theoretical approach allows to do this in so called Intent Specification (Leveson 2000) which complements the conventional methodology – it implements the "means – ends" hierarchy, where each level provides the intent ("why") information about the level below. The specification supports 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 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 enhanced as well as any analysis required when changes are proposed.
The levels are used to specify user requirements, environmental assumptions and safety constraints.

Especially useful feature of the intent specifications is strong support of traceability. The structure of the intent specification is such that each level above provides rationale for *why* decisions at the lower levels were made the way they were. To be able to follow this reasoning easily, traceability links are encouraged throughout the document. For example, each hazard would link, within the same level, to the safety design constraints that mitigate the hazard. Each safety constraint would then link down from level one to level two where design decisions comply with and enforce the constraint. Those design decisions would link down to level three, the subcomponent requirements, where the subcomponent requirements adhere to the system level design decisions. These links go from the highest level goals of the system all the way down to the code and operator manuals. By following these links, one follows the reasoning behind the system's behavior and can evaluate the safety of changes to the system.
We will review in the following four levels of the intent specification

### 8.1 Design and safety constraints – level 1

Requirements document what things the system should do. Constraints document things that the system should *not* do. Constraints provide limits on the space of possible designs within which the system will behave as desired. Intent specifications frequently divide constraints into those that are related to safety and those that are not. Safety constraints are design constraints motivated by safety concerns. It is fairly easy to derive safety constraints from hazards. For example, if a hazard for an autonomous mobile robot is having its arm extended while it is in motion, then the safety constraint could be written that the robot must not move while its arm is extended. Safety constraints link to the hazards that generated them and to the design decisions that enforce them.

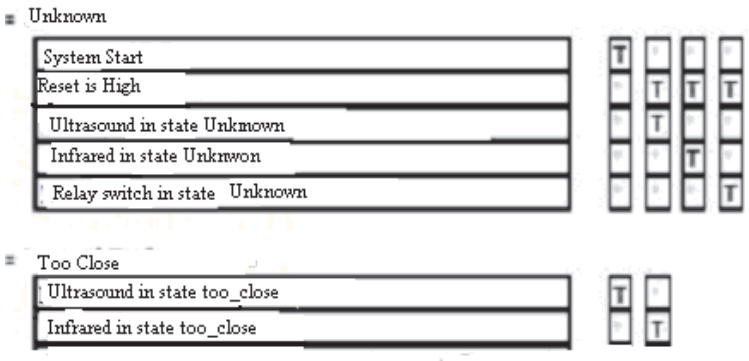
### 8.2 System design principles – level 2

The core of the level 2 of an intent specification is the set of design principles that specify how the design will satisfy the requirements documented in level 1 while not violating any design constraints. The functional design principles show the functional decomposition upon which the software logic is structured. It is useful to divide the robot control functionality into different operating modes (e.g. initialization, autonomy, operator, and safety). The functionality for each mode should be independent of the others. This feature allows the designers to change the internal logic of one mode without worrying about the effect the changes will have on the other modes.
The mode selection logic implements the mode transitions.

### 8.3 Black box specification – level 3

Level 3 is designed to provide the system designers with a complete set of tools with which to validate the specified requirements before implementation begins. Only black box (externally visible) behavior is included, i.e. the input/output function to be computed by the component. Black box models assist in the requirements review process by eliminating implementation details that do not affect external behavior and thus are not relevant in the validation of the requirements. For this purpose, a model of the robot control system is produced using formal specification language SpecTRM-RL (Leveson 2002).
Most model elements in SpecTRM-RL rely on AND/OR tables. The tables pictured below are part of the transitions for the “Distance” state in the obstacle avoidance behavior. AND/OR tables provide a very simple way to read the behavior of a component.

The first column of the table contains expressions that may be true or false. In the very first row of the table for the *Unknown* transition below, the expression is *System Start*. This expression is true at the instant the system is first started and false at all other times. The next row's expression is true when the *Reset* input has a value of *High* and false at other times.
When reading a table, the true or false value of this first column is matched against the values in subsequent columns. Subsequent columns must contain *T* indicating true, *F* for false, or * for don't care. If every row in the first column matches the values in some subsequent column, the table as a whole is true. If no column matches, then the table is not true. Another way of saying this is that rows are ANDed together and columns are ORed together
As an example, consider the *Unknown* transition below. If the table is true, then the system will transition to the state "distance to an obstacle" being unknown. This table can be read as saying that the distance transitions to unknown if the system is just starting, the system is being reset, or all the sensors are in the unknown state.
All the system state variables in a SpecTRM-RL model are required to have an "Unknown" value. A very common error found in requirements specifications and often associated with accidents is assuming that the computer always has an accurate (up-to-date) model of the controlled system. If input processing and feedback is disrupted for some reason (including temporary halting of the computer itself) however, the assumed controlled-system state may inaccurately reflect the real state. In SpecTRM-RL models, each state variable defaults to the unknown value at startup and returns to the unknown value after interruptions in processing or expected (and necessary) inputs are not received.




### 8.4 Physical and logical design – level 4

Level 4 describes the physical and logical designs of each subcomponent that allows them to meet the subcomponent requirements described at level 3. The component may be hardware, software, or some combination of the two.
Software Design Specifications can be represented in any design notation customarily used for software. Intent specifications can incorporate any desirable design notation, from formal predicate logic to UML diagrams.
Requirement specifications are mapped into design specifications of a selected robot control architecture. The mapping can be done in one of three ways:
**Uncoupled**: the mappings or functions from requirements to design principles are all one-to-one.

**Loosely coupled**: the mappings are one-to-many.
**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 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.

### 8.5 Determinism analysis

A system is nondeterministic when it may display one of several behaviors for the same sequence of inputs. Nondeterministic systems are very difficult to analyze and test because of the element of chance in their behavior. It is strongly desirable for a safety-critical system to behave in a deterministic fashion.
SpecTRM-RL models are models of the requirements for system components. Nondeterminism in the behavior of a SpecTRM-RL model indicates inconsistency in the requirements. Additional information must be added to the specification to collapse the case of nondeterminism down to one deterministic choice. The following is an example of nondeterminism in a SpecTRM-RL model.

= Operational

| | | | | |
|---|---|---|---|---|
| Reset | T | | | |
| Startup | | T | T | T |
| Valid sonar | | T | | |
| Valid infrared | | | T | |
| Valid relay switch | | | | T |

= Internal Fault Detected

| | | |
|---|---|---|
| Internal fault detected | T | |
| Startup | | T |
| Time since last entered Startup > 3 seconds | | T |

This example is taken from the robot control mode. According to the first table, the robot goes to the Operational mode any time the Reset button is pushed. According to the second table's first column, the robot transitions to Internal Fault Detected whenever it was already in Internal Fault Detected. This specification is ambiguous as to whether a reset command should put the system in an operational state once a fault has been detected.

### 8.6 Completeness analysis

A system is complete when there is a specified response for every sequence of inputs that might come into the system. Incomplete systems have combinations of system states and

inputs for which no response is specified. Incompletness occurs when there is no table in a state or mode definition that is true for some set of conditions. Additional information must be added to the specification to make one of the tables true.

## 9. Related methods

The closest to presented above approach to system safety is an industrial hazard analysis method called HAZard Operability analysis (HAZOP) developed for the British chemical industry in the 1950's. The goal of a HAZOP is to identify operational deviations from intended performance and study their impact on system safety (Soukas, 1988).
The HAZOP procedure is carried out by a HAZOP expert (the leader) and a team of system experts. The leader poses a battery of questions to the experts in an attempt to elicit potential system hazards. A HAZOP is potentially an exploratory analysis as neither potential faults nor hazards have been assumed beforehand. The HAZOP leader hypothesizes an abnormal condition and analysis proceeds in both directions, determining whether and how the condition is possible and what effects it has on the system.
The analysis is based on a systems theory model of accidents, in that it concentrates on the hazards that can result from components interaction, i.e. accidents are caused by deviations in component behavior.
HAZOP has several limitations. First, it is time- and labor –intensive, in large part due to its reliance on group discussions and manual analysis procedures. Second, HAZOP analyzes causes and effects with respect to deviations from expected behaviour, but it does not analyze whether the design, under normal operating conditions, yields expected behaviour or if the expected behaviour is what is desired.
A third limitations arises from the fact that HAZOP is a flow-based analysis. Deviations from within a component or processes are not expected directly; instead, deviation within a component (as well as human error or other environmental disturbance) is assumed to be manifested as a disturbed flow. A purely flow oriented approach may cause the analyst to neglect process-related malfunctions and hazards in favour of pipe-related causes and effects.
Because HAZOP concentrates on physical properties of the system (Souskas, 1988), it is not directly applicable to analyzing computer input and output.
There is a number of other manual methods for hazard analysis (e.g. Lear, 1993). All of these methods suffer from two weaknesses with respect to analyzing software. First, being manual techniques they depend on human understanding of the proposed software, which can be quite limited. Second, the manual techniques adhere to the HAZOP principle of identifying deviations in the connections, i.e., the computer inputs and outputs only. Accordingly, they do not provide guidance for following deviations into the control logic.

## 10. Conclusion

Knowledge of how to build safe and secure autonomous mobile robots is a prerequisite for their wide acceptance. In this chapter we have described relationship between robot autonomy and its safety and security. In contrast to traditional hazard analysis techniques used in electro-mechanical systems, we have introduced a technique based on system theory. We have outlined a methodology for building safe autonomous mobile robots based on hazard analysis and intent specification. The specification supports 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 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 enhanced as well as any analysis required when changes are proposed.

## 11. References

Alami et al (1998), An Architecture for Autonomy, International Journal of Robotics Research, April 1998.

Arkin, R. (1999) Behaviour-based Robotics,The MIT Press.

Asimov I. (1950), I Robot, Doubleday, Arkin, R. (1999), Behavior-Based Robotics, The MIT Press

Barchanski J.A.(2004), Safety and Security of Autonomous Robots Software, IAESTED International Conference on Advances in Computer Science and Technology, St. Thomas, US Virgin Islands, November 2004.

Barchanski, J.A. (2006), System-Theoretic Approach to Safety of Robot Control Architectures, Canadian Conference on Electrical and Computer Engineering, 6 - 10 May 2006, Ottawa.

Bertalanffy, L., (1969) General Systems Theory: Foundations, Development and Applications, G. Braziller, New York,

Chandler, A. et al (2001) Digging into Concurrency, Technical Report, Computing Department, Lancaster University,

Checkland, P., (1981), Systems Thinking. System Practice, John Wiley &Sons, NY Harel, D.,(1987) Statecharts: A Visual formalism for complex systems, Science of Computer Programming, 8:231-274,

Gordon, D. (2001). APT Agents: Agents that are adaptive, predictable, and timely. In Lecture Notes in Artificial Intelligence, Volume 1871. Springer-Verlag.

Jones, J.L.,(2004), Robot Programming, A Practical Guide, McGraw-Hill

Murphy, R., (1996), Biological and Cognitive Foundations of Intelligent Sensor Fusion, IEEE Transactions on Systems, Man and Cybernetics, vol. 26, No 1.

Murphy, R., (2000), Introduction to AI Robotics, The MIT Press.

Lear, J.,(1993), Computer hazard and operability studies, In Sydney University Chemical Engineering Association Symposium: Safety and Reliability of Process Control Systems, October 1993.

Leveson N. G.(1987), et al, Safety Analysis Using Petri Nets, IEEE Transaction on Software Engineering, March 1987.

Leveson, N.G., (1995), Safeware: System Safety and Computers, Addison Wesley,.

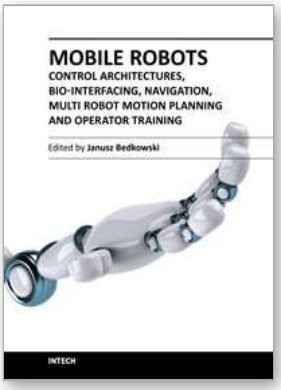
Leveson, N. G. (2000), Intent Specification: An Approach to Building Human-Centered Specifications, IEEE Trans. on Software Engineering, January 2000.

Leveson, N. G., (2002), Safety-critical Requirements Specifications using SpecTRM, Trans. on Soft. Engineering

Simmons, R., et al, (1999), Xavier: An Autonomous Mobile Robot on the Web, Robotics and Automation, Magazine,

Souskas, J.,(1988), The role of safety analysis in accident prevention, Accident Analysis and Prevention,

Stajano, F., (2002), Security for Ubiquitous Computing,Wiley,




**Mobile Robots - Control Architectures, Bio-Interfacing, Navigation, Multi Robot Motion Planning and Operator Training**

Edited by Dr. Janusz Będkowski

ISBN 978-953-307-842-7
Hard cover, 390 pages
**Publisher** InTech
**Published online** 02, December, 2011
**Published in print edition** December, 2011

The objective of this book is to cover advances of mobile robotics and related technologies applied for multi robot systems' design and development. Design of control system is a complex issue, requiring the application of information technologies to link the robots into a single network. Human robot interface becomes a demanding task, especially when we try to use sophisticated methods for brain signal processing. Generated electrophysiological signals can be used to command different devices, such as cars, wheelchair or even video games. A number of developments in navigation and path planning, including parallel programming, can be observed. Cooperative path planning, formation control of multi robotic agents, communication and distance measurement between agents are shown. Training of the mobile robot operators is very difficult task also because of several factors related to different task execution. The presented improvement is related to environment model generation based on autonomous mobile robot observations.



INTECH
open science | open minds

**InTech Europe**
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

**InTech China**
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821