Getting it Right the First Time: Predicted Performance Guarantees from the Analysis of Emergent Behavior in Autonomous and Semi-autonomous Systems

Ronald C. Arkin[a], Damian Lyons[b], Shu Jiang[a], Prem Nirmal[b], Munzir Zafar[a]
[a] School of Interactive Computing, Georgia Tech, Atlanta, GA 30332;
[b] Computer & Information Science, Fordham University, Bronx, NY 10458

ABSTRACT

A crucially important aspect for mission-critical robotic operations is ensuring as best as possible that an autonomous system be able to complete its task. In a project for the Defense Threat Reduction Agency (DTRA) we are developing methods to provide such guidance, specifically for counter-Weapons of Mass Destruction (C-WMD) missions. In this paper, we describe the scenarios under consideration, the performance measures and metrics being developed, and an outline of the mechanisms for providing performance guarantees.

Keywords: autonomous robots, performance guarantees, emergent behavior

1. INTRODUCTION

Robots are currently deployed in Iraq and Afghanistan against conventional explosive threats, e.g., improvised explosive devices (IEDS) but without the use of any significant level of autonomy. Weapons of Mass Destruction (WMDs), whether they by chemical, biological, or nuclear (CBN), obviously up the ante substantially and there is no tolerance for mistakes. Autonomy is increasingly demanded due to the large-scale hazard to human life and the need for a rapid response. However, to deploy autonomous systems effectively in such counter-WMD scenarios it is crucial to have a means of establishing performance guarantees for the systems.

The field of formal specification and verification of software systems has made impressive progress. However, leveraging these results to validate software for mobile robot systems has raised new challenges. In ongoing research for DTRA, we (1) introduce a concurrent, communicating process-based formal model for describing behavior-based mobile robot programs, as well as the environments in which the programs operate [Lyons et al 12b], and (2) a robot program development toolkit for robot software validation and verification functionality.

The software development environment is embedded into the *Missionlab* software, a comprehensive robot mission development, simulation, and execution environment. Once the mission has been created, the designer can choose to validate the program's behavior in a range of standard environments. The designer selects from a library of sensor and motor models that include a range of noise and uncertainty characteristics and can request the validation of the combination of robot program and environment for specific mission-critical properties of safeness, liveness or efficiency.

2. C-WMD SCENARIOS

Accompanying the proliferation of weapons of mass destruction, countering the threats of terrorist attacks using CMDs poses great challenges for any nation. The United States military believes that terrorist attacks using weapons of mass destruction is not a question of "if" but "when" [Dickson 99]. Robotic technology could play a useful role in the effort to counter future terrorist attacks as it has proved to do so in the battlefields of Iraq and Afghanistan. However, terrorist attacks with WMDs require different countermeasures from disarming conventional explosive devices; and the potential damage caused by WMDs can be far more devastating and lasting. A significant level of autonomy with stringent performance guarantees would likely be required for robots to be employed in counter-WMDs situations. Two shortcomings of the U.S. military's operational capabilities in countering WMDs identified

in [Dickson 99] can potentially be addressed with the robotic technology: 1) real-time detection and characterization of biological/chemical warfare agents, and 2) a capability to locate and disarm terrorist CBN areas.

In this section, we present a counter-WMD (C-WMD) scenario where a robot (or a team of robots) is tasked with the mission of searching and locating a biological WMD inside a building. WMDs can be categorized into three basic types: chemical, biological, and nuclear (CBN). However, biological weapons may be more likely to be used in terrorist attacks than its counterparts [Dickson 99]. First, biological agents are easier to produce because the technology is dual-use and widely available. "Anyone who makes home-brewed beer can make anthrax," said Brigadier General John Doesburg, the former Director of the Joint Program Office for Biological Defense [Dickson 99]. Just last year, in 2011, the U.S. National Institutes of Health asked scientists not to publish their research on a deadly form of bird flu virus, the H5N1 avian flu, in fear of it to be used for terrorism, although now it appears it will be published in full [Greenfield-Boycs 12]. Second, it is easier to deliver attacks with biological warfare agents. The well-publicized anthrax attack in the United States in 2001 was carried out, allegedly by a single individual, by sending out envelopes of anthrax spores through the postal service.

Consider a hypothetical scenario where a terrorist group has created a new mutation of the H5N1 avian bird flu that can be passed easily among humans, and the group is planning to use the virus during the ball drop of 2013 New Year's celebration in a square in a major U.S. City, which is attended by approximately 1 million people from all over the world every year. Fortunately, the Central Intelligence Agency (CIA) intercepted the intelligence about the terrorist group's plan and acted on it immediately by soliciting the help of the Defense Threat Reduction Agency (DTRA), which is highly specialized in combating the threat of WMD. The intercepted information suggests the likely location of the H5N1 virus is in the basement of a specific building in that area, but the exact location of the virus within the basement is unknown. DTRA decides to deploy its team of C-WMD robots into the target building to search, locate, characterize, and neutralize the biohazard.

The mission assigned to the team of C-WMD robots requires them to first drive up a ramp to get onto the loading dock at the back of the possibly contaminated building, and then enter the basement through a double-sided door. Once inside the building, C-WMD robots are required to autonomously travel along the hallway, entering each room, while searching for any signature of the biohazard. If a robot finds and confirms the presence of a biological weapon, it sends an alert message to the operator at the base-station indicating that the biohazard has been found, the biohazard's location, and any additional information the robot obtained about the biohazard from its sensors. The robot then waits for further instructions from the operator regarding how to proceed next (e.g., to return to the base-station or to neutralize the biohazard).

For this paper, a simplified version of the biohazard search scenario is simulated using *MissionLab*[1] (a mission specification and robot control system developed for tasking teams of robots [MacKenzie et al 97, Georgia Tech 07]) and illustrated using a Pioneer robot (Fig. 1) conducted in the basement of the building in which the Georgia Tech Mobile Robot Laboratory is located. The robot started off at the bottom of the ramp that leads up to the loading dock, where the basement entrance is located. It then traveled up the ramp and entered the basement through the double-sided door. The robot then traveled along the hallway, until it reached the end of the hallway where the Mobile Robot Lab is located, where we had placed a "biohazard", which is represented by a red container with biohazard sign on it. The robot then entered the Mobile Robot Lab, and found the biohazard. Navigation was conducted autonomously through this area, without additional human intervention.

[1] *MissionLab* is freely available for research and educational purposes at: http://www.cc.gatech.edu/ai/robot-lab/research/*MissionLab*/

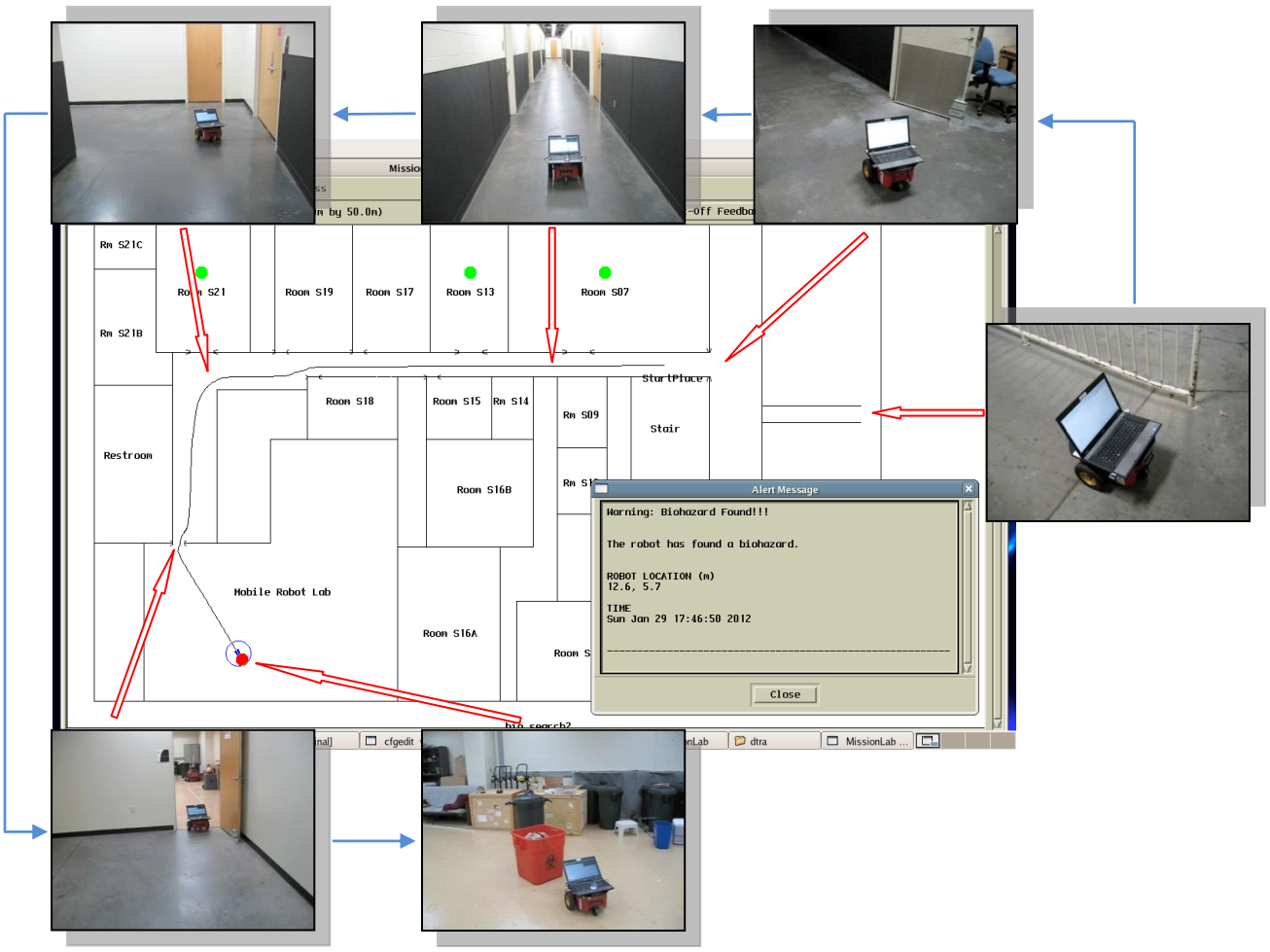


Figure 1: Biohazard Search Scenario. Still shots of the robot execution are overlaid on top of the simulation run. In the simulation, the black trail behind the robot is the path the robot just traveled, and the solid red circle represents the biohazard.

3. UNDERSTANDING PERFORMANCE GUARANTEES

A performance guarantee is the quantification of the ability of a specified mission to complete its task in the given environment. The verification approach presented in this paper attempts to provide us with such guarantees. Specifically this means is that the verification module (Section 3) will provide a number or a distribution function to the mission designer. This number expresses the probability that the specified mission representation, referred to as a finite state acceptor (FSA) in *MissionLab*, will be successful in completing its mission in the specified environment with the specified robot. A pertinent question to ask then is, what if the robot hardware itself fails? What if the internals of the robot operating system fail? What if the communication link to the base station fails? What if the operator(s) in charge of the mission providing cues to the semi-autonomous robots make an error? The mission FSA specified can be guaranteed to succeed only when these other factors operate within accepted and guaranteed levels of performance. Thus, the overall success of the actual mission on the ground is not equivalent to the performance guarantees solely based on the mission FSA, but is rather a more comprehensive function including it. We call this function "mission effectiveness." Since the aim of our research is to estimate the reliability of the actual mission, these factors need to be incorporated into the expression for mission effectiveness.

The phrase "mission effectiveness" has been used interchangeably with "system effectiveness" and was first introduced in early 60's [Tillman et al 80] to describe the capability of military and space systems to successfully complete assigned missions. [Tillman et al 80] provides a comprehensive review of the literature on system

effectiveness as published in defense journals. System effectiveness is typically modeled as a function of probabilities such as reliability, availability, repairability, maintainability etc. of the system. A weakness in even the most complete models is that they cannot provide the most accurate estimates, as the underlying probabilities may not be accurate [Tillman et al 80]. Also, the system effectiveness literature in the past has focused more on determining an aggregate number to represent the average effectiveness of a system over a range of mission types and environments. This is due to the need of characterizing a system for customer considerations, as contractors require these effectiveness measures to specify system capabilities. In addition, as pointed out in [Soban and Mavris 01], "As such, the system effectiveness concept was applied to a single component or tool that itself was defined as the system."

What makes our application different is that we are not aiming to put an aggregate assessment on a system or component for a range of environments or mission types. Rather, we are interested in putting an assessment in context, for a specific mission to be carried out in a specific environment. This assessment will change from mission to mission, and from environment to environment. Also, the mission involves a team composed of robots and humans in the case of semi-autonomous systems, and comprises the overall system under consideration. Nonetheless, the concepts and models presented by [Tillman et al. 80, Lie et al. 84] are applicable, in part, to our framework.

3.1 Mission Effectiveness

We adopt the definition of mission effectiveness [Lie84] as "the probability of successfully completing an assigned mission." The system in consideration is comprised of the following components:

1. Mission Specified (*MissionLab* FSA)
2. Robot Hardware
3. Robot Software
4. Communication Link
5. Human Operator (the operator and/or mission designer)

Mission effectiveness is expressed as:

$$ME = Pr\{U \leq u_{max} | S\} \, Pr\{ S \}$$

where:

ME	Mission Effectiveness
U	Duration of mission completion
u_{max}	Maximum allowable mission duration
S	Operating state is successfully maintained by the hardware and other components of the system

The first term $Pr\{U \leq u_{max} | S\}$ is the conditional probability that the specified mission FSA being verified, will complete its operation successfully in the required time, given the other components on which the operation is dependent are working properly. This probability characterizes the mission FSA. The second term $Pr\{ S \}$ is the probability that the other components in the system do not fail and captures the reliability of the rest of the system on which the FSA is dependent for its execution.

The second term in the expression is a joint probability, combining the reliabilities of individual components within the system. If these components are assumed to be independent, then this joint probability will be a simple product of the individual component reliabilities. The National Institute of Standards and Technology (NIST) has provided a list of standard metrics and testbeds for the evaluation of specific robot hardware capabilities [Jacoff and Messina 11]. The reliability of a simple/complex communication network has also been studied extensively. As for the human component, various models for human effectiveness in man-machine systems have been explored in [Lee88]. Human reliability is, in general, a function of training, selection, experience, motivation and stress/workload.

The second term, thus, puts an upper bound on mission effectiveness. The more we can know about the subsystems, the easier it is to estimate this upper bound. The first term then enables us to zero in on the best possible estimate of

the overall mission, by formally analyzing the mission specified. This process as applied to the specific mission, is explained in the next section.

4. VERIFICATION

This project's aim is to incorporate a process-based formal model for describing behavior-based mobile robot programs and their environments into *MissionLab* in support of automated verification. It is evident that as the missions that robots take on become more complex, so does the robot programming. Poor characterizations of the capabilities of the robot and incomplete modes of the environment have caused the downfall of many autonomous systems [Braman 2007]. The verification module is used to establish that the design of the robot program under consideration possesses certain properties when carried out in the specified environment. Such properties can be elementary, e.g. the system never reaches a deadlock state. The basis of system verification is the performance criterion, which can be for example, a safety property ('nothing bad happens'), a liveness property ('the goal is eventually reached'), mission efficiency, or some combination of them. The system is considered "correct" whenever it satisfies all performance criteria. Hence, correctness is relative to a specification, which includes the environment in which the program is to be executed, and is not an absolute property of a system [Baier and Katoen 2008]. When dealing with complex systems, the output of verification is not always a succinct, yes/no solution. We aim to facilitate a range of outputs, both discrete and non-discrete by providing a performance report rather than a yes/no answer.

4.1 Verification Module

Figure 2 depicts how the verification module is integrated into *MissionLab*. The inputs to the verification module are the behavioral program (specified in an intermediate language referred to as the Configuration Network Language (CNL) [Georgia Tech 07]), sensor, robot, and environment models, and all performance criteria specified by the operator. The CNL file constituting the operator's control program is translated into the language of our verification framework, which is called PARS (*Process Algebra for Robot Schemas*). The semantics of PARS is a specific kind of automaton, developed by Steenstrup, Arbib and Maines [Steenstrup, Arbib 83] to represent concurrent communicating processes, called a *Port Automaton*.

Referring to Figure 2, the operator selects or assembles a configuration of the robot(s), sensor(s), and environment models for the mission from the model library. All of these models have previously been processed using an offline pre-processor. The offline pre-processor converts the model (robot, sensor, environment) and the performance criteria into an internal tabular format that can be quickly linked with the robot program. Each model library and performance criteria need only to be pre-processed once, offline, and then can be re-used many times as needed. The same pre-processing step is applied to the robot control program. However, the mission phase may be reiterated several times until acceptable performance is achieved.

4.2 Linker

The robot control program is composed in *MissionLab* using a configuration editor (CfgEdit), independent of the details of the platform on which it will run and the environment in which it will operate. However, for verification purposes, the robot and environment models must be declared and linked correctly to the program. The linker establishes a **system network**: a concurrent composition of controller, robot, sensor, and environment. Figure 3 shows the connection interface between controller and robot/sensor/environment. The controller acts on the environment through the robot model, and receives information about the environment from the sensor model. In a simple environment model, where the robot is the only entity in the environment, the robot model might pass the information directly to the sensor model. In reality, the robot is always part of the environment, and so are the sensors. The robot communicates with its environment by several methods; for example, the robot may displace an obstacle by making contact with it – this is a change in environment. The environment may also displace the robot, thus the communication between robot and environment is a two-way communication. Similarly, the environment may interfere with the robot's sensory equipment by making contact with it. For example, an obstacle such as a tree in the environment may damage the robots sonar module or camera orientation upon contact. Acknowledging such connections between robot, sensor, and environment are imperative towards system verification.

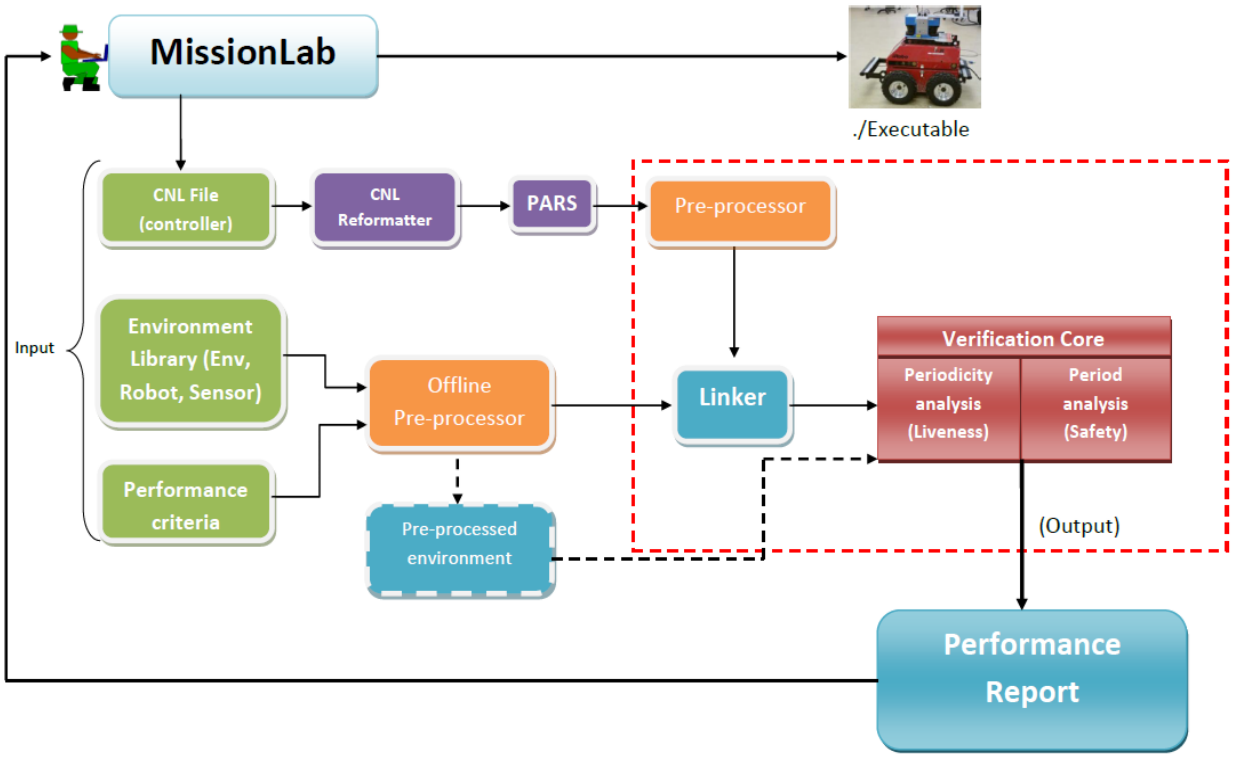


Figure 2. Verification module

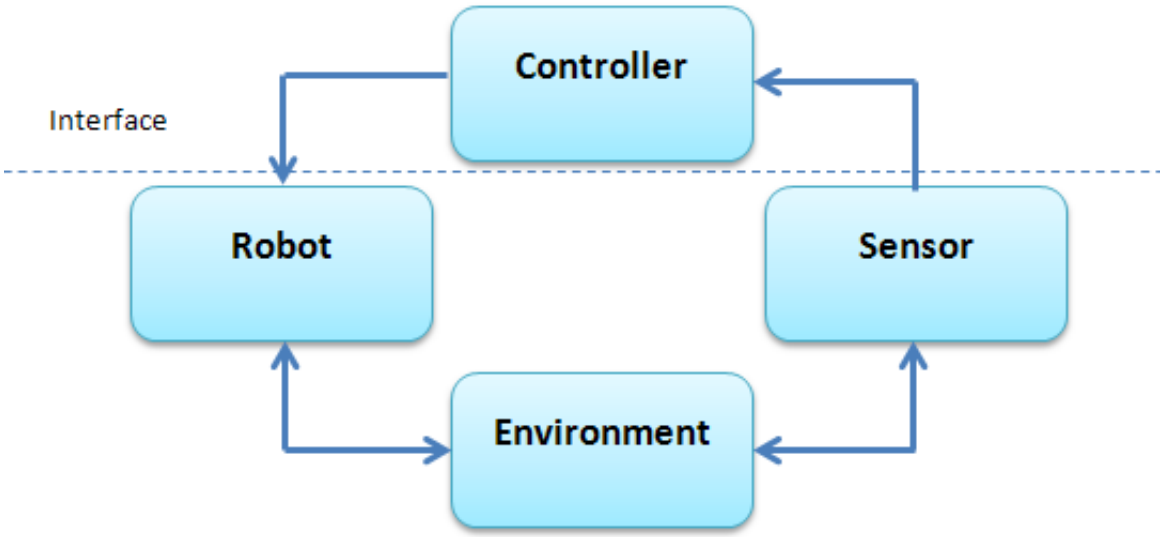


Figure 3. System Connectivity

4.3 Verification Core

Figure 4 depicts the verification core. By analysis, the combination of robot program and robot/environment model is rephrased into a repetitive component – the system period – and a component that controls the repetition; see [Lyons et al 12b] for more details on this approach. Verification proceeds by attempting to map the robot and environment model onto the performance criteria. This mapping is established by two methods, described below.

- *Periodicity Analysis*: This type of verification is applied to liveness properties. A liveness property states that 'good things will eventually happen,' thus it is necessary to analyze several iterations of the system for verification, using *flow functions*. By use of flow functions, the variables' values are projected from one repetition of the period to the next. As a simple example, consider a robot travelling through an empty space. The robot is assigned a velocity v for time t. A recursive process is used to model the robots position. The position of the robot is updated at each iteration of the recursive process – this update in position is modeled using a flow function. Let the robot's position be p in the first iteration and $p + vt$ in the next iteration. Then, the flow function mapping the two position parameters is $f(p) = p + vt$.

- *Period Analysis*: To verify that the system exhibits a safety property, the verification module abstracts the system down to one iteration of the system period. As mentioned earlier, a safety property states that 'nothing bad happens.' Thus, it is necessary to verify that the safety property is guaranteed to hold for each iteration of the system period. An example of a safety property is that the robot not collide with sensed obstacles. In each period, if an obstacle is sensed, the robot must perform an action in response that results in avoidance. The system is represented as a network of recursive processes, hence verifying the safety property in one iteration suffices to show that the safety property holds in general.

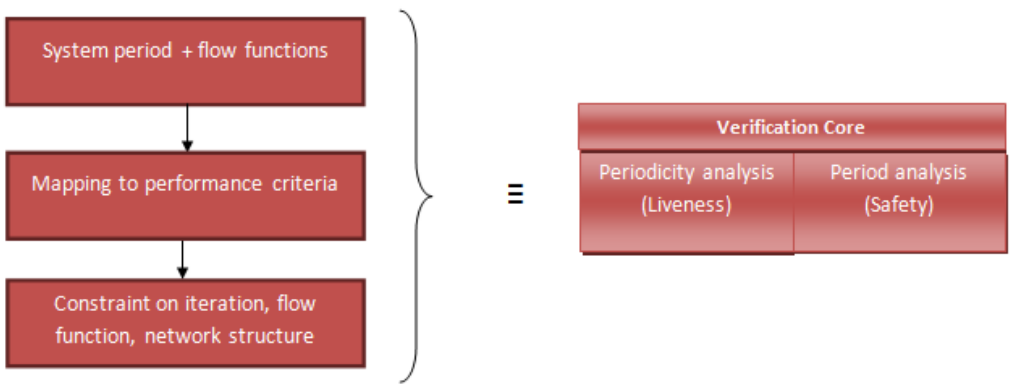


Figure 4. Verification Core

4.4 Output

An output of "yes" or "no" if a program meets or fails to meet a performance criterion respectively, while useful to a designer, does not help in improving the designed control program (mission). The objective is rather to generate a performance report that includes information to help the designer improve the performance of the robot program. For a simple scenario where a simple solution suffices as a performance guarantee, the verification module may return a clear go/no-go decision based on analysis of the system. However, for a more complex system exhibiting probabilistic behavior, a simple yes-or-no answer is not practicable for the operator. A yes-or-no answer would imply determining the reachability of the system. However, the state-space of robotic systems exhibiting uncertainty is expected to typically be of exponential complexity. Therefore, verifying the mission based on reachability would not prove to be an efficient method, nor useful to the operator.

For example, after the verification analysis of a robot/environment system exhibiting uncertainty and noise related to pose and speed estimation, the verification module will return a time and space distribution as a performance report. Consider a robot programmed to move back-and-forth between point A and point B. The robot and environment models exhibit noise and uncertainty such as from uneven terrain, which cause the robot's wheels to slip. When we take into account the factors that affect the robot's behavior, an estimate of the robot's pose at the end of n iterations can be provided in terms of a spatial distribution. This additional information is a more useful tool for the designer to improve the performance of the program than a simple yes/no answer. Robotic systems exhibit high levels of

uncertainty related to position estimation. Therefore, it is highly unlikely that the robot will be at a precise position upon execution of the program. Thus, a yes/no would be worthless in as a performance report to the designer.

Dynamic environments and sensor failure increase the complexity of autonomous systems and increase the number of ways that robots can fail [Braman 2007]. A dependency analysis will also be part of the performance report. The dependency report identifies and ranks actions and communications in the program based on their effect on the mission performance. For example, the operator may find it useful to know how the robot's velocity affects the mission's outcome. Increasing the robot's velocity may decrease the likelihood of mission safety, but may increase the likelihood of mission completion within a certain time limit. What if the robot exhibits a sensor failure? The failure of a sensor module may draw a fine line between success and failure of a mission – in such scenarios the operator must know which hardware components may potentially compromise mission success. Focusing on the failure of hardware such as sensors and motors, we aim to expedite system verification to assess cause-consequence relations between component faults and hazards that may occur during the lifetime of a system. More details on the system verification methods employed in our approach can be found in [Lyons et al 12a, Lyons et al 12b].

5. SOFTWARE DEVELOPMENT ENVIRONMENT

The robot software verification environment is embedded into the *MissionLab* software mission specification system, a comprehensive robot mission development, simulation, and execution environment [Mackenzie et al 97, Endo et al 04]. To provide performance guarantees for the robot control software, a new software verification step is added before in-the-field mission execution to predict the robot's performance via the analysis of the underlying behaviors of the autonomous systems. Overt behaviors can emerge from interaction with the environment that were not initially designed or intended by the robot programmer. Some of these emergent behaviors might be undesirable or can even result in failure to the overall mission. Criticality and timeliness requirements of C-WMD missions could necessitate the need for a significant a priori guarantee on the mission's success before undertaking it. A retry may not be feasible, so the robot must get it right the first time. Thus, a rigorous analysis of the robot software is necessary before robot deployment to assist in ensuring success.

5.1 System Architecture

The architecture of the underlying mission specification/verification system is shown in Figure 5. The details of using *MisisonLab* to design robot missions can be found in [MacKenzie97, Georgia Tech 07]. The basic design process starts with the mission developer creating a visual robot program (Fig. 6a) in *CfgEdit*, the graphical programming interface of the *MissionLab*, where a robot control program is entered as a finite state automaton (FSA). When the designer is satisfied with the controller design, he/she can proceed to generate the robot executable through a series of compilations as shown in Figure 5. The mission control program can then be evaluated within the simulation environment provided by *MissionLab* or directly executed on actual robot platforms. However, the designer can exploit the software verification system described in this paper to assist in ensuring successful mission completion and to provide a level of performance guarantee to the operator prior to deployment. The verification module checks the robot program using a combination of environment, robot, and sensor models and provides the operator with at a minimum a "yes" or "no" regarding whether the robot program satisfies all of the mission requirements. Additionally, the verification module can generate a list of unsatisfied constraints, which is useful for the designer to refine her robot control program in the case of an unsatisfactory performance evaluation. The software verification system forms an iterative design feedback loop for writing robot control programs (Fig. 5).

5.2 CNL to PARS Translation

The specified robot mission representation (initially in the Configuration Description Language (CDL)) is translated to the verification language, PARS (Process Algebra for Robot Schemas [Lyons et al 12b), from its CNL (Configuration Network Language) representation for use in the verification module (Figure 5). Generation of CNL is one of the intermediate steps before the robot program is finally compiled into a robot executable in *MissionLab* (Fig. 5). CNL represents the robot program as a network of nodes, where nodes are behaviors and they are connected to other nodes via their input and output ports to form the complete robot program, an assemblage/network of behaviors.

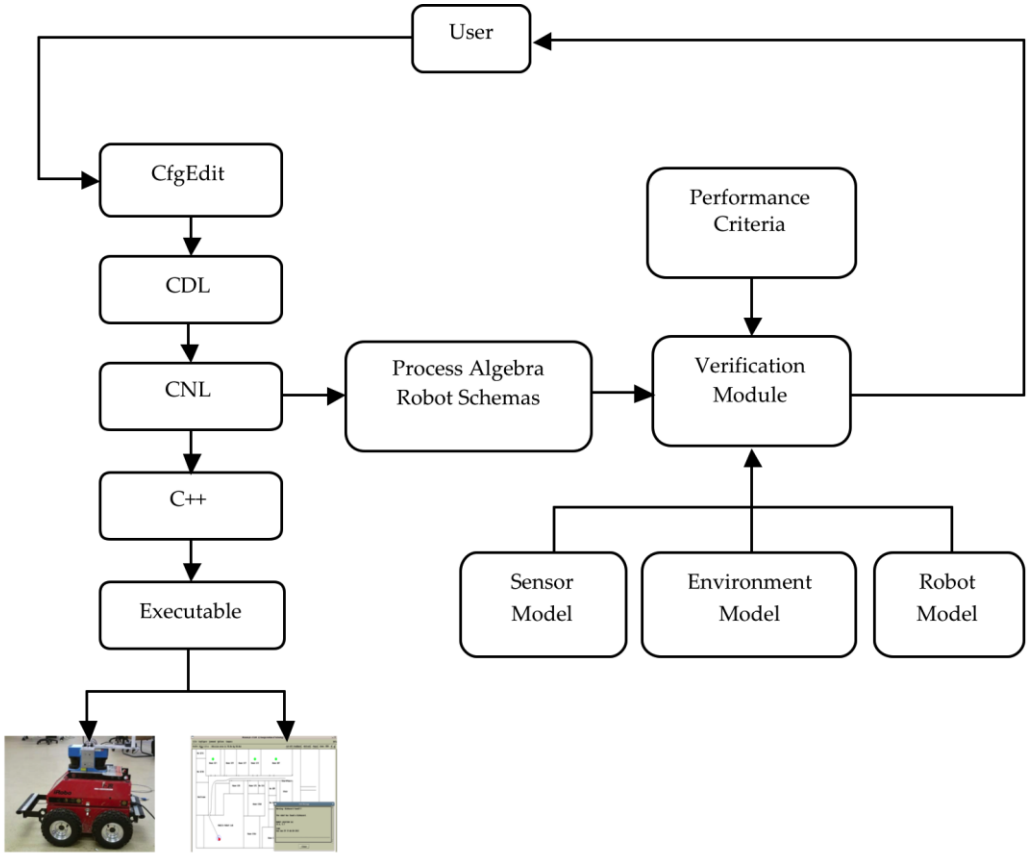
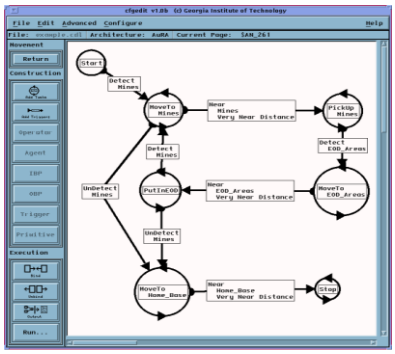


Figure 5: Software Development Architecture



FSA1(in1,in2)(out1) =

((IS_AT_GOAL<g1,5.0>(in1)(res) # RPEQ<true>(res)) # PASS<in1,out1>);
((IS_AT_GOAL<g2,5.0>(in1)(res) # RPEQ<true>(res)) # PASS<in2,out1>);

FSA1.

a) CfgEdit b) CNL c) PARS

Figure 6: Robot program, from FSA to PARS

5.3 Environment Model Library

What makes the verification of robot software challenging and different from traditional computer software verification is that a robot has to interact with the environment. The environment is incorporated into the verification system by providing it with an explicit environment model library, from which the operator can select the environment that the robot program would be verified with (Fig. 7). Each model can be further customized by the operator to reflect the specific environment parameters.

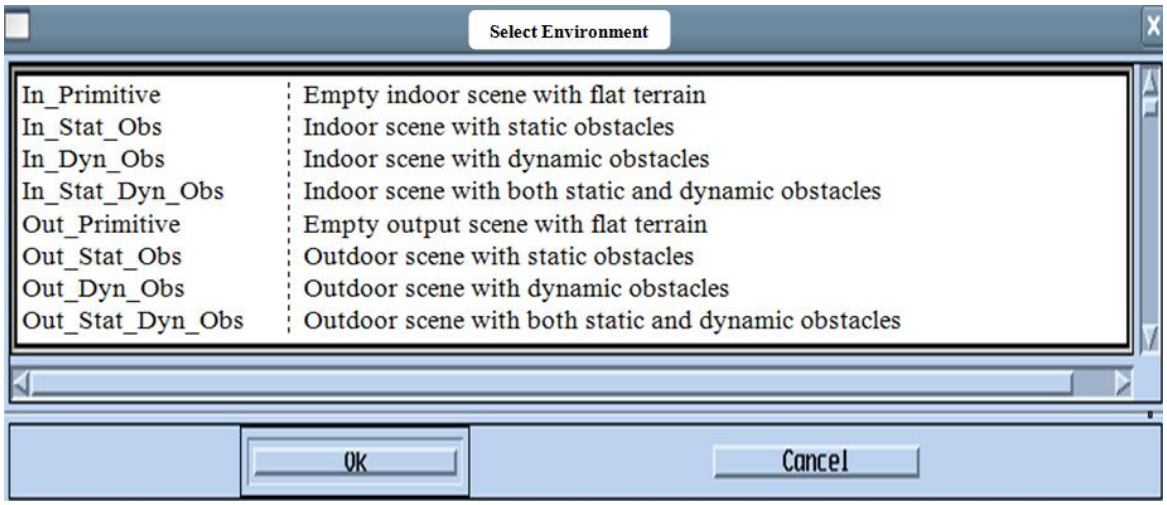


Figure 7: Example of the Environment Model Library

The environment model library is built up by first categorizing the environment into two basic categories: indoor and outdoor environments. Each category starts with a simple primitive model and additional models with increasing complexity are added to the category by building upon the primitive model. The motivation of this type of library construction is to reuse existing simple models as the building blocks for more complex models. For example, In_Stat_Obs is constructed by adding static obstacles to the In_Primitive model (Fig. 7).

The structure of the environmental model similar to the one used by [Hasegawa 91] was adopted, which is composed of a geometric description and a physical description. A geometric description includes dimensions, positions, and orientations of environmental entities that make up the environment. A physical description includes an environmental entity's weight, center of gravity, surface friction and other distinct properties. For the biohazard search scenario presented earlier (Fig. 1), an environment model similar to In_Stat_Obs can be constructed. The geometric description of the environment would include the 3D layout of the building basement, which can be obtained from building floor plan or CAD drawings. The geometric description would also include dimensions and positions of objects (e.g., trashcan) within the environment. Layout of the building can be obtained from the floor plan and drawing of the building. Physical description would include properties such as surface friction of the floor and objects' weights.

5.4 Robot Model Library

To do any job properly, the right tool has to be used. The same principle applies when selecting a robot for a mission. The mechanical configuration of a robot defines its ability to act. For a given environment and mission, some robots might succeed where others might fail. To take into account a robot's embodiment in the physical world, a robot model library is provided for the operator to choose the desired robot platform to verify the control software (Fig. 8).

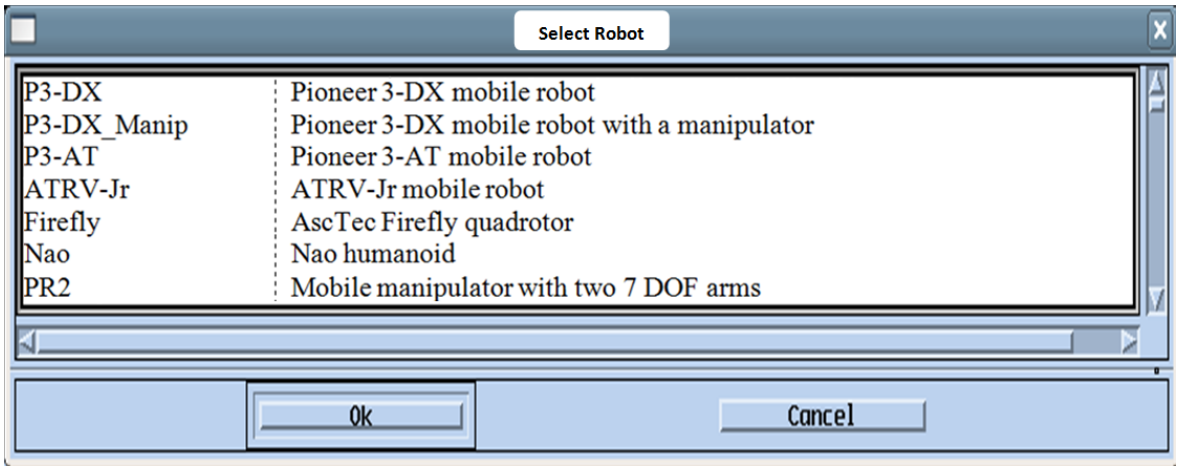


Figure 8: Example of the Robot Model Library

The robot model library starts with a family of standard robot platforms. Additional robot platforms or customization of the standard models can be added to expand the model library. A robot model can be described by the robot's kinematic model, which can be derived from its datasheet or obtained from the literature. A robot's kinematic model can be written as

$$\dot{x} = f(x, u, e) \tag{1}$$

Where x is the state of the robot such as robot pose, u is the control input such as wheel speed and steering angle, and e is the motion error. The motion error can be assumed to be Gaussian and characterized by its mean and covariance matrix. If the kinematic model of a robot can be approximated as a linear system, then the equation in (1) can be simplified to

$$\dot{x} = Ax + Bu + e \tag{2}$$

where A is the system evolution matrix and B dictates how the control inputs affect the system.

Besides the kinematic model, other information about the robot such as battery life, tire properties, and weight are also encapsulated within the robot model. An example of the robot model data structure is shown in Figure 9 for a Pioneer 2-DX robot. Pioneer 2-DX is a differential drive wheeled mobile robot (WMR) with two motorized wheels and one caster wheel (Fig. 10). This robot platform was used in the biohazard search scenario presented earlier (Fig. 1). The uncertainty in robot motion is assumed to be Gaussian noise and is reflected in the data structure with the error mean and error covariance matrix. The most signification contribution to uncertainty in robot motion is from the slippage between the robot's wheels and the floor, which was incorporated in the robot model as a robot parameter. Battery life is also important because a robot would fail a mission if it did not have sufficient energy to complete it.

```
struct Kinematics {
   array[3][3]     A;
   vector          currentState;
   vector          nextState;
   double          u;        // Control input
   vector          B;
   double          e;        // Gaussian error in state transition
   double          mean;     //  mean of error
   array[3][3]     covar;    // covariance matrix of error
};
```

```
struct RobotParameters {
   double          Payload;
   double          Weight;
   double          WheelRadius;
   vector          Dimensions;
   double          WheelFriction;
   double          WheelSlip;
   double          MaxBatteryLife;
}
```

Figure 9: Example model data structure for Pioneer 2-DX robot

Figure 10: Pioneer 2-DX

5.5 Sensor Model Library

Uncertainty doesn't exist in the environment. It is the byproduct of our sensors' inability to fully observe the world as it is presented. While perfect sensor doesn't exist, uncertainty can be minimized by using appropriate sensors. In a similar fashion to previous model libraries, a sensor model library for a suite of standard robot sensors is provided for the verification module, (Fig. 11). Each sensor model is represented by a measurement model:

$$y = f(x, n) \tag{3}$$

where x is the state of the system being observed (e.g., robot pose or obstacle location) and n is the measurement noise. Similar to motion error, the measurement noise can be assumed to be Gaussian and characterized by its mean and covariance matrix. Linear measurement model can also be used when it is appropriate

$$y = Fx + n \tag{4}$$

where F maps the system state to the sensor output.

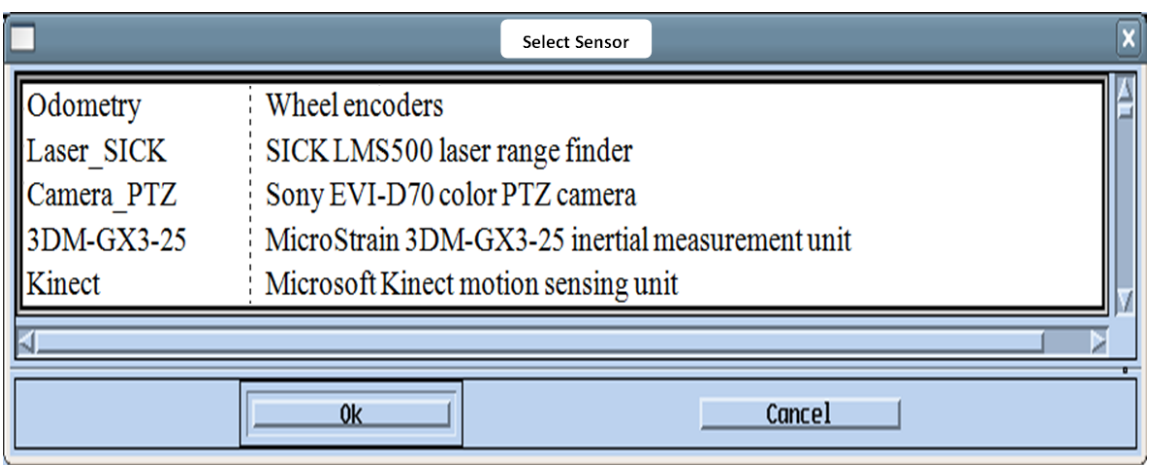


Figure 11: Example of the Sensor Model Library

A sensor model also includes specific physical properties of an individual sensor. The model for a particular sensor can be derived from its manufacturer datasheet or the literature. For instance, [Ye 02] characterized a 2-D laser scanner, specifically the SICK LMS 200 (Fig. 12a). The properties that are modeled for the laser scanner include its angular resolution, data rate, range, and error (Fig. 12b).

	Angular resolution = 1°, 0.5°, or 0.25° Data Transfer Rate = 500 Kbaud Max Range = 8 m Error = 15 mm Standard deviations bound = (0.8, 9.1)
a)	b)

Figure 12: (a) SICK LMS-200 laser scanner (b) LMS-200 Characteristics from [Ye 02]

5.6 Iterative Mission Development

The robot software development system facilitates robot mission development that verifies the correctness of the control software within the content of the robot platform, sensors in the environment. Recall the biohazard search scenario presented earlier in Section 2. Suppose the operator would like to verify the robot program she designed in *MissionLab* for a Pioneer 2-DX robot with sonar sensors and a SICK laser scanner to conduct the biohazard search mission in the contaminated building. After the operator compiled her program, she would start the verification phase by first automatically translating the robot program into the verification language, PARS. Then the operator would select the library model for the environment shown in Figure 1. The operator would then choose the Pioneer 2-DX robot from the robot model library, and sonar sensors and SICK LMS 200 from the sensor model library. The verifier then outputs whether the robot system satisfies all mission constraints as specified by the operator. If the robot system failed to meet mission requirement, the operator could inspect the verifier's output (e.g., unmet constraints) to refine her robot program or change the robot or sensor hardware. For example, if the verifier determined the robot system failed to meet the time requirement (e.g., too slow), the operator could simply increase the robot's speed or perhaps use a faster robot..

6. SUMMARY

This paper presents an architectural overview of a mission verification system with the goal of providing performance guarantees regarding missions intended for use in C-WMD operations. Performance metrics, an integrative architecture, and a motivating scenario have been presented.

Additional details on this project for the Defense Threat Reduction Agency can be found in our other publications on the subject [Lyons et al 12a, Lyons et al 12b]. What distinguishes this research is the explicit acknowledgment of the environment and hardware interactions with the control program in the verification phase, not simply verifying the control code by itself. Further the ability of the output of the verification system to provide guidance to an operator to improve mission performance is also designed into the system from the onset is also a novel contribution.

Future work involves the completion of the implementation, and testing on single and multiple robots, first in a laboratory testbed and then in conjunction with the NIST testbeds [Jacoff and Messina 11].

ACKNOWLEDGMENTS

This work was supported by the Defense Threat Reduction Agency, Basic Research Award # HDTRA1-11-1-0038, to Georgia Tech.

REFERENCES

Baier, C and Katoen, J-P, Principles of Model Checking, MIT Press, 026202649X (2008).

Braman, J.M.B., Murray, R.M., and Wagner, D.A, “Safety verification of a fault tolerant reconfigurable autonomous goal-based robotic control system”, Proc. IROS 2007, San Diego CA (2007).

Dickson, L.E., “The Military Role in Countering Terrorist Use of Weapons of Mass Destruction,” The Counterproliferation Paper Series, (1999).

Endo, Y., MacKenzie, D., and Arkin, R.C.,"Usability Evaluation of High-level User Assistance for Robot Mission Specification", IEEE Transactions on Systems, Man, and Cybernetics, 34(2),168-180 (2004).

Georgia Tech Mobile Robotics Laboratory, MissionLab User Manual, Version 7.0, http://www.cc.gatech.edu/aimosaic/robot-lab/research/MissionLab/mlab_manual-7.0.pdf, (2007).

Greenfield-Boyce, N., “WHO Panel Supports Publication of Bird Flu Details, Eventually”, NPR, http://www.npr.org/blogs/health/2012/02/17/147053720/who-panel-supports-publication-of-bird-flu-details-eventually, last accessed 2/29/2012.

Hasegawa, T., Suehiro, T., and Takase, K.; “A Robot System for Unstructured Environments Based on an Environment Model and Manipulation Skills”, IEEE International Conference on Robotics and Automation, Sacramento, CA, (1991).

Jacoff, A. and Messina, E.; “Standard Test Methods For Response Robots”, ASTM E54.08.01 Intelligent Systems Division, National Institute of Standards and Technology (2011).

Lee, K.W., Tillman, F.A., and Higgins, J.J.; “A literature survey of the human reliability component in a man-machine system”, IEEE Transactions on Reliability, 37(1), (1988).

Lie, Chang Hoon, Kuo, Way, Tillman, Frank A., and Hwang, C. L.; “Mission Effectiveness Model for A System with Several Mission Types”, IEEE Transactions on Reliability, R-33(4), (1984).

Lyons, D., Arkin, R., Fox, S., Shu, J., Nirmal, P., and Zafar, M., "Characterizing Performance Guarantees for Multiagent, Real-Time Systems operating in Noisy and Uncertain Environments", Performance Metrics for Intelligent Systems (Permis 2012), College Park, MD, (2012a).

Lyons, D., Arkin, R., Fox, S., Shu, J., Nirmal, P., and Shu, J., “A System Architecture for the Design of Autonomous Robot Missions with Performance Guarantees”, in submission (2012b).

MacKenzie, D., Arkin, R.C., and Cameron, R., “Multiagent Mission Specification and Execution”, Autonomous Robots, 4(1), 29-52 (1997).

Soban, D.S. and Mavris, D. N., “The Need for a Military System Effectiveness Framework: The System of Systems Approach”, 1st AIAA, Aircraft, Technology Integration, and Operations Forum, Los Angeles, CA, (2001).

Steenstrup, M., Arbib, M.A., Manes, E.G., “Port Automata and the Algebra of Concurrent Processes”, *JCSS* 27(1), 29-50 (1983).

Tillman, F.A.; Hwang, C.L.; Kuo, Way; “System Effectiveness Models: An Annotated Bibliography”, IEEE Transactions on Reliability, R-29(4), (1980).

Ye, C. and Borenstein, J.,“Characterization of a 2-D Laser Scanner for Mobile Robot Obstacle Negotiation”, IEEE International Conference on Robotics & Automation, Washington, DC, (2002).