\

# A Software Tool for the Design of Critical Robot Missions with Performance Guarantees

D M. Lyons[a*], R.C. Arkin[b], P. Nirmal[a], S. Jiang[b], T-M. Liu[a]

[a]*Fordham University, Bronx NY*
[b]*Georgia Institute of Technology, Atlanta GA*

**Abstract**

Deploying a robot as part of a counter-weapons of mass destruction mission demands that the robotic software operates with high assurance. A unique feature of robotic software development is the need to perform predictably in a physical environment that may only be poorly characterized in advance. In this paper, we present an approach to building high assurance software for robot missions carried out in uncertain environments. The software development framework and the verification algorithm, VIPARS, are described in detail. Results are presented for missions including motion and sensing uncertainty, interaction with obstacles, and the use of sensors to guide behavior.


Selection and/or peer-review under responsibility of Georgia Institute of Technology.
Keywords: performance guarantees; verification; mobile robotics.

## 1. Introduction

We have developed a novel formal framework for the analysis of probabilistic and behavior-based robotic algorithms intended for use in both single and multi-robot applications [1][2]. The resulting methods and tools provide a basis for establishing performance guarantees of these algorithms in complex real-world environmental situations, which is crucial for the critical real-time real-world needs of search, analysis, and recovery tasks. Specifically considered here are high assurance mission needs for application in the domain of counter-weapons of mass destruction (C-WMD) where the robot moves and navigates in an uncertain environment.

The robot software verification problem shares many characteristics with real-time program verification, but is different in that the environment in which the robot operates may vary widely and be only partially known. The issue of interaction with a dynamic and uncertain environment raises many difficulties for generating high assurance software, chief among these being the need to handle noisy sensors and actuators, probabilistic software and environment information, and time-critical operation,.

In the next section, we introduce the application domain of robotic software for C-WMD missions, its key characteristics and its points of difficulty for establishing performance guarantees. Section 3 describes how a mission is specified in the *MissionLab* robot mission specification toolset [3] and how the software is associated with robot, sensor, and environment models for the purpose of providing performance guarantees. The domain-specific verification model using PARS (Process Algebra for Robot Schemas [1]) and algorithm, VIPARS (Verification in PARS), is reviewed in Section 4. Section 5 presents the results of verifying performance guarantees for several example missions. We conclude in Section 6 with a discussion of our research results and outline our next steps.

* Corresponding author. Tel.: +1-718-817-4485; fax: +1-718-817-4488.
*E-mail address*: dlyons@fordham.edu.

## 2. Critical Robot Missions

The motivation for this research is straightforward:

- Time-critical situations are a reality when dealing with the potential impact of WMDs. Robotic systems can and should play an important role in dealing with an imminent threat from these agents.
- WMDs, whether they be 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.

The goal of this research is to provide effective methods to deal with WMDs using robotic technology that get it right the first time, and perhaps the only time, to do so. Through the use of formal methods to establish provable properties of intelligent robotic systems this goal is within reach.

C-WMD missions pose unique challenges for robotic systems. Typical target scenarios involve identification, search, recognition and reporting of chemical, biological, or radiological targets. This requires the ability of the robot to move in a predictable manner through an environment that may be only poorly characterized in advance. While we consider both indoor and outdoor test locations, there is an initial focus on indoor applications as seen in later sections of this paper.

**Figure 1:** Mission tasking by a human operator for a search task for a single robot area search mission.

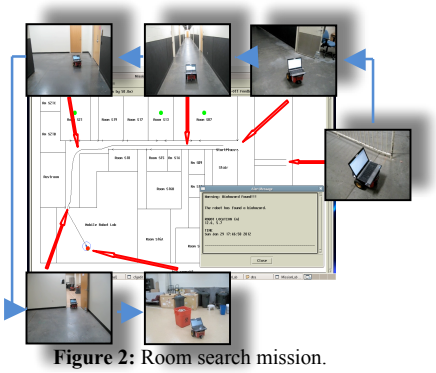

**Figure 2:** Room search mission.

The many significant problems associated with the verification of the mission software used for these robotic applications that relate predominantly to the coupling of the agent to its environment – i.e., it is not simply verifying the performance of the software but rather the software controlling the actuators and interpreting the sensors relative to the world's characteristics. These manifold problems include:

1. Modern robotic software of this sort is characterized by emergent behavior [4] – i.e., there is a holistic effect produced by the individual components due to the unpredicted arrival of sensory events and locomotion irregularities. How to deal with such unpredictability due to the integration of the environment (and the means to interact with it – the sensors and actuators) compounds the difficulty of verification and requires the use of accurate sensor and actuator models.
2. Probabilistic software [5] is now also commonplace in modern robotic software. This is often used to generate the internal world models for simultaneous localization and mapping (SLAM) – letting the robot know what the world looks like and where it is relative to it. Quantifying the performance of such random systems poses a different set of challenges.
3. The actual execution of C-WMD missions may include not only a single autonomous robot but often engages a semi-autonomous mode where a human operator provides real-time guidance or advice during the execution if communication conditions permits. Further, multiple robots are used in many situations to provide distributed and ideally coordinated sensing and acting, which requires models of the performance of the other agents during execution, which may be difficult to ascertain.
4. Uncertainty is rampant in this environment: in sensing, in acting, and in the a priori assumptions and models of the world made during the generation of the mission software prior to execution.
5. This is further complicated by the possible failure of hardware components, i.e., the robot and sensors. This leads us to use a mission effectiveness metric ("the probability of successfully completing an assigned mission" [6]) to characterize overall performance [7].

While this paper focuses primarily on simple single autonomous robotic missions (Figure 1), the intent is to extend the framework described herein over the next year to multi-robot scenarios coupled with human supervision during execution in both indoor and outdoor search for WMDs.

One mission example may be a robot approaching and entering a building following a series of waypoints (Figure 1). No GPS is available when inside. The building floor may be slippery but its characteristics may be well known in advance due to existing human intelligence. The mission ends when a WMD is encountered by the robot and that knowledge transmitted back to the operator.

A second example is more qualitative (Figure 2): the robot needs to visit a series of rooms (not metric waypoints) proceeding in and out of each room's door as required, without full knowledge of the layout in advance. Both the floor's traction conditions and the door locations are not well known a priori, requiring sensors to find them.

## 3. Specifying Robot Missions

To ensure autonomous robots' success in critical missions where failure could be catastrophic, we introduce a pre-deployment performance analysis tool to provide a guarantee on mission success. This verification tool is built upon *MissionLab* [3], a robot mission specification toolset created based on the principles of behavior-based robotics [4]. The high-level software architecture for robot mission specification coupled with verification is shown in Figure 3.

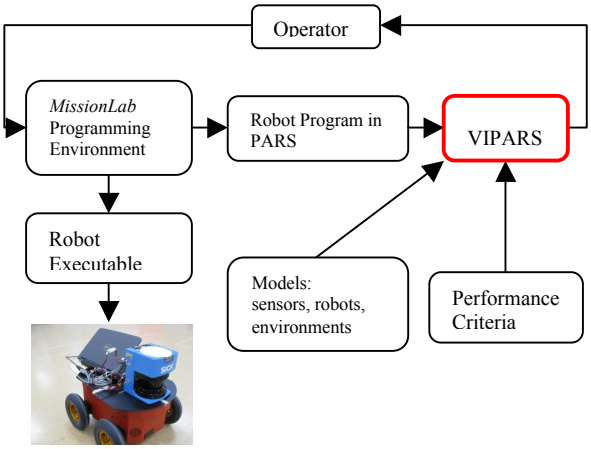


**Figure 3:** *MissionLab* robot mission specification toolset with VIPARS verification module

### *3.1 Design with Guarantee*

The core of *MissionLab* is a library of primitive behaviors. Complex behaviors are created by assembling these primitives together. The user front-end of the *MissionLab* system is a usability-tested visual programming interface for specifying robot missions [8][9]. With this graphical programming tool, the robot operator creates the robot controller as a finite state automaton (FSA). Figure 4 shows the FSA created in *MissionLab* for the waypoints-based navigation example presented earlier. The FSA consists of four *GoTo* behaviors with each behavior having a different goal location based on the given waypoints.

When the mission operator is satisfied with her design of the robot controller, she can generate the executable to be executed on a robot or multiple robots. However, for time-critical missions such as counter weapons of mass destruction or urban search and rescue (USAR), additional mission verification should be carried out to assure the robot's predicted performance before its deployment to prevent catastrophic failures.

Robots interact with their environment and they behave differently in different environments. Unintended behavior can emerge from robots' these interactions that were not initially envisioned or programmed by the robot operator. Thus, in order to establish a performance guarantee, the robot program must be associated with specific robot, sensor(s), and environment models. The operator's robot program is then automatically translated from CNL (*MissionLab's* program encoding) [3] to PARS (Process Algebra for Robot Schemas [1]), the language understood by the VIPARS verifier used in this research. The performance criteria and models of the robot, sensors, and environment are also specified by the operator as inputs for the verifier.

Continuing with the waypoint navigation example, the operator might choose a Pioneer 3-AT robot (Figure 5) to follow the designated waypoints in a relatively open 30 meters x 30 meters room with a tile floor. The robot is equipped with a SICK laser range scanner, a gyro, and wheel encoders for odometry. The laser scanner measures the robot's distance to objects in the environment and is used for obstacle avoidance. Wheel encoders count the wheel revolutions and are used to estimate the position of the robot. The gyro measures the heading of the robot. The

performance criteria depend on the objectives of the mission. For our example, the criteria might be the robot has to move within at least 0.2 meters radius of each waypoint and cannot approach any obstacle closer than 0.5 meters.

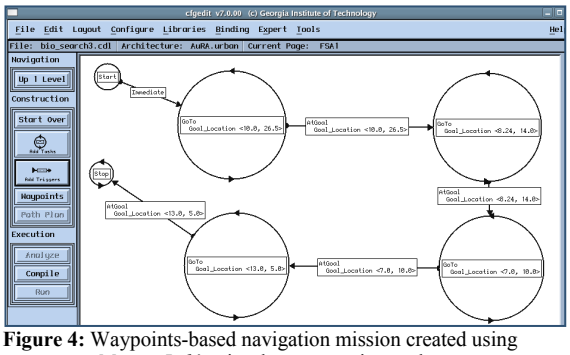


**Figure 4:** Waypoints-based navigation mission created using *MissionLab*'s visual programming tool.

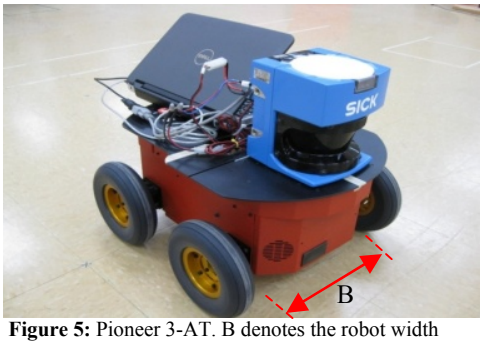


**Figure 5:** Pioneer 3-AT. B denotes the robot width

The verifier then provides the predicted performance of the mission to the operator. The performance guarantee is quantified as a probability distribution that describes the likelihood of a robot successfully accomplishing the mission [7]. Besides this numeric data, the verifier should also provide meaningful information to the operator, e.g., which part of the robot system (e.g., sensors or robot program) cause the most severe degradation in the predicted performance. The verifier effectively forms an iterative design loop whereby the operator can make changes to their initial design based on verifier feedback until satisfactory performance is guaranteed. Even changing out the robot platform is an option if the chosen robot cannot get the job done. For example, if the environment contains oil patches on the floor, then a Pioneer 3-AT might not be able to follow the waypoints correctly due to significant slippage between its wheels and the oily floor. The operator then might choose a different robot platform that provides better traction on oil than the Pioneer 3-AT is capable of.

### *3.2 Uncertainty*

Robots have to operate in the real world that is full of uncertainty, which remains a challenging research problem in robotics. Researchers have proposed several different methods for tackling this problem, e.g., probabilistic robotics [5]. Uncertainties exist in a robot's actions due to imperfect actuation, in sensors due to noise, and in the environment due to unknown dynamics. For example, when a robot is commanded to move forward, it rarely travels in a perfectly straight line due to slippage between its wheels and the surface. How much the robot deviates from the straight path depends on the terrain and cannot be fully determined ahead of time. Thus, in order to get an accurate performance guarantee, uncertainties need to be included in the robot, sensors, and environment models for the verifier.

For this research, we introduce uncertainty into the robot model. For the waypoint navigation example, this uncertainty represents the error between the commanded position and the robot's actual position. It is modeled by a covariance matrix $\Sigma$, which propagates along the path as the robot travels from one waypoint to the other. The kinematic model of the skid-steered Pioneer 3-AT robot is given by:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}\frac{r}{2} * cos(\theta) & \frac{r}{2} * \cos(\theta)\\ \frac{r}{2} * \sin(\theta) & \frac{r}{2} * sin(\theta)\\ -\frac{r}{\alpha * B} & \frac{r}{\alpha * B}\end{bmatrix}\begin{bmatrix}\omega_l\\ \omega_r\end{bmatrix} \qquad (1)$$

where $\dot{x}$ *and* $\dot{y}$ are the velocities of the robot in the lateral direction and the longitudinal direction, respectively, $\dot{\theta}$ is the robot angular velocity; $\omega_l$ and $\omega_r$ are the angular velocities of the left and right wheels, respectively; B is the robot width (Figure 5); $r$ is the wheel radius; and $\alpha$ is a function of the instantaneous center of rotations (ICRs) and is terrain-dependent, e.g., $\alpha = 1.5$ for vinyl laboratory surface [10]. The position of the robot in the world frame can be derived from (1) by integration over time, which would be associated with a covariance matrix $\Sigma$ to represent the uncertainty in this position estimation.

## 4. Verification

The field of automated software verification has produced several effective software tools. The most well known among these include SPIN [11], originally designed to verify communication protocols, and PRISM [12] for verification of probabilistic systems. SPIN targets deterministic and nondeterministic systems. Nondeterminism represents the potential occurrence any of a number of possible events, without preference as to which might occur. In contrast, a probabilistic approach represents this situation when the probability of all events are equal, but can also represent any foreknowledge we happen to believe or have measured about the situation. For this reason, we prefer to be able to represent the probability of events for our application. PRISM can reason about a wide variety of probabilistic systems such as Discrete and Continuous Time Markov Chains (DTMC, CTMC) and Markov Decision Processes (MDPs). However, PRISM captures probabilistic actions by guarded actions labeled with discrete probabilities. There is a role for discrete probabilities in our application; for example, we may be able to express the probability that the door to a room will be open in a building the robot is searching. However, we also need to be able to represent the distribution of probabilities over a parameter range, for example, the location of an obstacle in space is more usefully represented as a probability distribution over that region of space rather than as a discrete probability that it is at a specific point P. For our application therefore, we prefer to represent uncertainty related to sensing and actuation as distributions over state (e.g., location and velocity). Indeed, such a-priori information may be readily available from equipment manufacturers or from empirical data [14]. PRISM does support CTMC with exponential distributions on time, but this is the only distribution it supports. While this distribution is widely-used for event occurrence, it is not appropriate in general for spatial variables, where a normal distribution is more generally applicable

In model-checking, the problem state space is the Cartesian product of program variables of the program [11][13], and reachability in this space is the principle paradigm. The combinatorial size of the space is a well-known problem in model checking [13]. In our case, this problem is made worse by the inclusion of uncertain robot, sensor and environment models, rendering the combined state-space of controller (program) and environment model prohibitively large to use reachability as a verification paradigm. A standard approach to state space explosion is to search for state regularities that can be leveraged to reduce the size of the space; for example the *assume-guarantee* approach to modularizing a system, or the use of *fairness conditions* to eliminate undesirable states from analysis [13]. In [1], we introduced a novel state regularity appropriate to robotics and possibly other real-time domains with uncertainty and complex environment interactions: the *System Period*. This is described in more detail in [1] which focuses on that aspect of our work. The System Period allows us to avoid inspecting the entire state-space of the combined controller and environment model, and just to inspect the state-space of the System period. We leverage the complexity reduction this allows. Our main tool is process algebra [15], which allows process descriptions to be transformed to investigate issues such as process equivalence as well as liveness, safety, and deadlock. Below we review our earlier work, briefly introducing the algebra, showing example missions and environments, followed by presenting the verification approach.

### *4.1 PARS*

PARS is a process algebra for representing robot programs and environments for analysis and verification of robot behavior. The semantics of a process in PARS is a extended port-automaton [16], an automaton equipped with communication ports, message transmission and reception, and extended with duration and partitioned end-states (success/stop and fail/abort).

A process P with initial parameter values *u1,u2,…* input ports/connections *i1,i2,…,* output ports/connections *o1,o2,…* and final result values *v1,v2,…* is written as:

$$\mathbf{P}_{u1,u2,\ldots}\ (i1,i2,\ldots)\ (o1,o2,\ldots)\ \langle v1,v2,\ldots\rangle$$

For brevity, the parts of a process description that are empty are typically omitted. Processes that are defined only in terms of a port-automaton are referred to as basic processes, the atomic units from which programs are built (Table 1).

Non basic processes are defined in terms of compositions of other processes. For example a process **T** that inputs a value on port *c1* and then outputs it on port c2 is defined:

$$\mathbf{T} = \mathbf{In}_{c1}\langle x\rangle\ ;\ \mathbf{Out}_{c2,x}$$

For simplicity, we define sequential composition to cover both sequence and condition operations (Table 2).

A tail-recursive (TR) process is written as:

$$\mathbf{T} = \mathbf{P}\ ;\ \mathbf{T} \tag{2}$$

This describes a process that repeats **P** continually until it aborts. This corresponds to the looping construct in a general purpose programming language. Any language that implements sequence, condition and loop constructs is in fact sufficient to represent any program [17]; thus, we can be confident that PARS can represent any program.

***Table 1:*** *Examples of Basic Processes*

| Process | Stop | Abort |
|---|---|---|
| $\mathbf{Delay}_t$ | After time t | If forced |
| $\mathbf{Ran}_{\Phi}\langle v\rangle$ | returns a random sample $v$ from a distribution $\Phi$. | If forced |
| $\mathbf{In}_c\langle y\rangle$ , $\mathbf{Out}_{c,x}$ | perform input and output, respectively, on port $c$ | If forced |
| $\mathbf{Eq}_{a,b}$ , $\mathbf{Neq}_{a,b}$ , $\mathbf{Gtr}_{a,b}$, etc., | $a=b$, $a\neq b$, $a>b$, etc., | Otherwise |

***Table 2:*** *Process Composition Operations*

| | |
|---|---|
| **T = P ; Q** | behaves like P until P terminates. If P stops, then T continues to behave like Q, otherwise T aborts. |
| **T = P \| Q** | behaves like P and Q concurrently and stops when both terminate. |
| **T = P # Q** | behaves like P and Q concurrently but stops when either terminates (aborting the other). |

### *4.2 Missions, Environments and Goals*

A very common kind of robot program is one that directs the robot to visit a series of waypoints (Figure 4), e.g.,

$$\mathbf{Mission}_{w,i} = \mathbf{MoveTo}_{w(i)} ; \mathbf{Neq}_{i,n}; \mathbf{Mission}_{w,i+1}$$

The $\mathbf{MoveTo}_{w(i)}$ is a process that directs the robot to go to the $i^{th}$ waypoint $w(i)$ where the waypoints specify locations along the search path and where there are $n$ waypoints. We can write a very simple controller as:

$$\mathbf{MoveTo}_g = \mathbf{In}_p\langle r\rangle ; \mathbf{Neq}_{r,g} ; \mathbf{Out}_{v,\, s(g-r)} ; \mathbf{MoveTo}_g \tag{3}$$

where $s(d)$ maps a position difference to an appropriate velocity vector, pulling the robot towards the goal using the potential field [18] approach until the goal is reached. The port $p$ provides position information from the robot mechanism and the port $v$ directs the velocity value to the robot mechanism. The robot mechanism and its behavior *have* to be included in this network for verification to be possible.

We represent the uncertainty in robot motion using random variables from an a-priori characterized probability distribution (see Section 5). It is important to note that we are as interested in spatial probability distributions as we are in the temporal distributions typically analyzed for queuing and network protocol examples. The environment model $\mathbf{Env}_r$ below includes both sources of uncertainty as random variables:

$$\begin{aligned} \mathbf{Env}_r \;\; &= (\mathbf{Delay}_t \,\#\, \mathbf{Odo}_r \,\#\, \mathbf{At}_r) ; \\ &\quad \mathbf{Ran}_{\Theta}\langle z\rangle \; ; \mathbf{In}_v\langle u\rangle ; \; \mathbf{Env}_{r+(u+z)t} \\ \mathbf{Odo}_r \;\; &= \mathbf{Ran}_{\Phi}\langle e\rangle \; ; \mathbf{Out}_{p,r+e} \; ; \mathbf{Odo}_r \end{aligned} \tag{4}$$

The process $\mathbf{At}_r$ represents the robot at location $r$. The process **Odo** (short for Odometry sensor) transmits the robot location in a loop until terminated by the **Delay**. In our results, we use a multivariate normal distribution for these distributions. The actuator noise is characterized by the distribution $\Theta \sim N(\mu_s, \sigma_s)$ and sensor noise by $\Phi \sim N(\mu_m, \sigma_m)$.

Stochastic properties that can be modeled in PARS include probabilistically delayed enabling/disabling and synchronization as well as random variable values. As a robot moves through a search area, the terrain may vary. Prior knowledge of this variation may be limited. For example, the existence and duration of difficult terrain patches (represented by the process **Rough**) in such an uncertain environment can be captured in PARS as:

$$\begin{aligned} \mathbf{TerGen} = \; &(\mathbf{Ran}_{\Psi}\langle t\rangle \mid \mathbf{Ran}_{\Phi}\langle d\rangle) ; \\ &\mathbf{Delay}_t ; (\mathbf{Delay}_d \,\#\, \mathbf{Rough}) \, . \end{aligned}$$

Where $\Psi \sim Exp(\lambda)$ and $\Phi \sim N(\mu, \sigma)$ (this is a combination referred to as a *severity* [19]). This framework can represent previously unseen or unknown objects and events, and the adaptive/learning algorithms to handle them [5], as long as the potential for these events is written in the form above.

A designer may wish to know if a specified system completes a task (*liveness*). In our approach, both the system and property to be verified are specified in PARS as process networks. The two process networks are compared and analyzed, and if the specified system can be shown *equivalent* to the property, we report success.

### *4.3 The System Period*

A behavior-based robot interacting with its environment [4] will respond to a specific set of environmental percepts as programmed by its behaviors. In this situation, which we refer to as a behavioral state because the robot continually responds to the fixed set of sensory percepts, a periodic regularity is induced in the combined state-space of program and environment. Once a percept is responded to, the robot may return to this behavioral state or move to another that

handles a different set of percepts. However, the essence of the behavioral state is this potential to repeatedly handle a specific set of percepts.

The repetitive element in PARS is tail-recursion (TR):

$$\mathbf{T}_u = \mathbf{P}_u ; \mathbf{T}_{f(u)} = \mathbf{P}^n_{f^{n-1}(u)}; \mathbf{T}_{f^n(u)} \quad n>0,\ f^0(u)=u \tag{5}$$

where $\mathbf{P}^n$ denotes $\mathbf{P^{n-1}};\mathbf{P}$. $\mathbf{T}$ offers a repeated interaction with its environment, but now the effect of previous events to the *ith* iteration is captured by the sequence of values of $f^i(u)$. We will refer to $f$ as a *parameter-flow function*: a mapping $f : \mathbb{D}^m \rightarrow \mathbb{D}^m$ that relates the values of $m$ parameters in the *nth* and *(n+1)th* iterations of the period **P.** The use of flow functions allows us to handle the issue of real-valued variables ($\mathbb{D} = \mathbb{R}$) transformed by processes, and this feature is one of the principal motivations in using tail recursion as a loop construct.

When we analyze a robot program operating in a given environment, we analyze a concurrent, communicating composition of the program and the environment. Consider a set of TR process equations **P1**, **P2**, ..., **Pm** that form a system **Sys** through concurrent composition:

$$\mathbf{Sys} = \mathbf{P1} \mid \mathbf{P2} \mid \ldots \mid \mathbf{Pm}$$

If this is an implementation of a behavior-based program, then our earlier reasoning about behavior states prompts us to ask, can this **Sys** also be rewritten in a TR form? That is, under what conditions can we develop an expression for the *period of the entire system* in terms of the *periods of its component processes*? If we can do this, then we can write a parameter-flow function for the entire system in terms of the parameter-flow functions of the components by *just* inspecting the communications that transpire in *one* system period; a big reduction in complexity for verification.

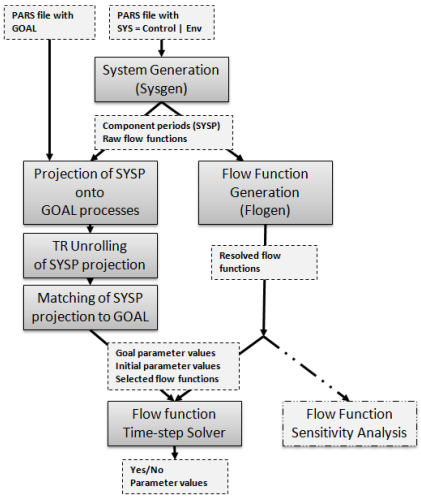


**Figure 6:** VIPARS Algorithm (Verification in PARS)

**Sysgen**( $PS = \{p_1, \ldots p_n\}$ ) // component periods

1. For each $p_i \in PS$
2. Generate port IO signature $io_i$
3. $ts_i \leftarrow$ first port operation in $io_i$
4. $TS \leftarrow \{ ts_i \}$ , $Cycles = \varnothing$
5. $cm(ts)$ is the network connection for $ts$
6. While $TS \neq \varnothing$
7. Cycles $\leftarrow$ Cycles $\cup$ $\{TS\}$
8. Repeat Select $ts_j \in TS$
9. While $cm(ts_j) \notin TS \wedge$ *not tried all TS yet*
10. If $cm(ts_j) \in TS$ // *matched*
11. Move $ts_i$ and $ts_j$ to next operation
12. If none left in $ts$
13. $TS \leftarrow TS/ts$
14. Reset $ts$; $US \leftarrow US \cup \{ ts \}$
15. else // *unmatched?*
16. If $cm( ts_j) = ts_i \in US$ // *unroll, ie consider* $p_i = p_i ; p_i$
17. Move $ts_i$ and $ts_j$ to next operation
18. $TS \leftarrow TS \cup \{ts_i\}$
20. If $TS \in Cycles$ // *no valid unrolling exists*
19. else // *communication deadlock*!

**Figure 7:** System Period Generation Algorithm

### *4.4 Verification Algorithms*

We have developed a verification approach that leverages the system period concept. The verification algorithm VIPARS, is shown in Figure 6. The principal verification steps are:

1. The system period, SYSP, and parameter-flow functions for the combination of controller and environment model are generated.
2. SYSP is projected onto processes in the performance specification network G, SYSP$\downarrow$G.
3. G is matched to SYSP$\downarrow$G, and the constraints on the process parameter values determined.
4. The set of recurrent flow functions is solved for these constraints.

The algorithm used to generate the system period is shown in Figure 7 (Sysgen). Because port IO operations do not have fan-in or fan-out, the complexity of Sysgen is determined in the worst case by the number of component process (*n)* and port operations *(m)* in each, times the number of unrolling (*n* in the worst case*)* as $O(n^2 \times m)$.

The projection of a process network to a set of processes S can be accomplished by traversing the network and mapping any process not in S to the empty process. This is a process abstraction that reveals a portion of the network. The matching between performance specification and projected system period is a structural equivalence. If both networks have the same composition structure with the same process arguments, then the networks are considered

equivalent. The parameters of processes in the performance specification network must then equal the parameters of the equivalent processes in the system period. This establishes the initial and final constraints under which the system of recurrent parameter-flow functions must be solved. We currently solve the system of flow functions numerically using an upper bound on time. Our approach is to use the best available tools to address this problem (e.g. PURRS[a]).

## 5. Results

The *MissonLab*/VIPARS system is in active development but can already handle some useful robot programming examples. In this section we present the results from the verification of a waypoint mission. In each case the robot controller attempts to move the robot from a point P0 to a point G, the first waypoint. The condition being verified is that the robot is at the point *G* after some time $t<T_{max}$ with probability $p>P_{min}$ (and see [1] for more details of verification conditions in PARS). The first set of results, subsection 5.1, illustrates what the output from the VIPARS verification module looks like. The second set of results, in subsection 5.2, shows the more complex output when the environment contains obstacles to the motion. The final results, section 5.3, are validation results that compare the accuracy of prediction to experimental measurements of robot motion.

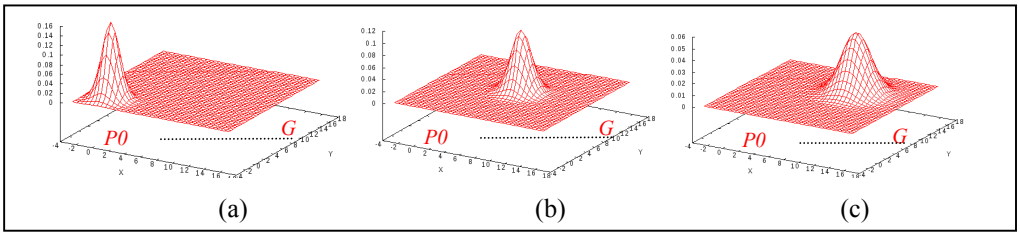


**Figure 8:** Three snapshots of the robot position distribution from P0 (a) to G(c). The output of VIPARS is the distribution (c).

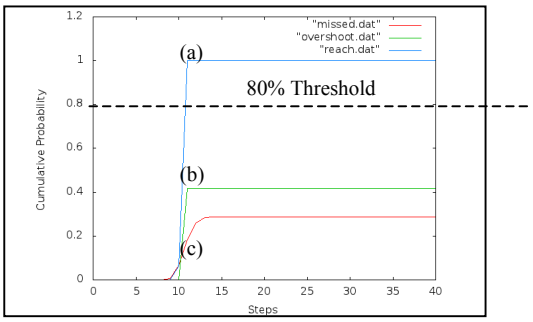


**Figure 9:** Cumulative probability of being at G versus time; successful verification (a), unsuccessful (b) and (c).

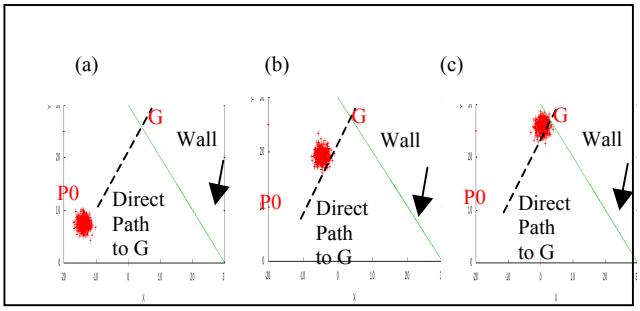


**Figure 10:** Inital(a), interim(b) and final (c) position distribution (wall)

### *5.1 Verification of Motion to a Goal Point G.*

The controller in (3) and probabilistic environment in (4) were verified against the liveness condition that the robot is at the point *G* after some time $t<T_{max}=1000s$ with probability $p>P_{min}=0.8$. The motion error distribution was measured on a Pioneer 3AT robot for various velocities and distances on an indoor, lab-floor surface and used to construct a velocity-dependent normal distribution $\Theta$ for motion error. The odometry sensing error $\Phi$ was not measured separately in this example. Figure 8 shows several snapshots of the position distribution as it approaches the goal location G. The probability of being at G increases with each iteration of the system period until it reaches the threshold $P_{min}=0.8$. This $P_{min}$ threshold is provided by the designer as part of the performance specification. Figure 9(a) shows a graph of the increase of the probability of the robot of being at G against time. Since the robot position is represented at all times as a spatial distribution, the cumulative probability of being at G rises monotonically as the robot approaches G; the initial low probabilities representing the cases when the robot motion error is so small that the robot arrives at the goal relatively quickly. The verification module returns the position distribution where the probability of having arrived at G exceeds the threshold if it exists, and signals that the liveness condition holds.

If the probability can never be above the threshold $P_{min}$ before $T_{max}$ then the verification returns that the liveness condition can not hold. A system can fail to meet the performance criterion if the robot controller is logically defective (Figure 9(b)) of course, but also if the motion and sensing noise is too large (Figure 9(c)). Currently VIPARS only returns the distribution information and verification condition holds/doesn't hold status to *MissionLab*.

---

[a] http://www.cs.unipr.it/purrs.

In the future we plan to exploit the connectivity information used in Sysgen to perform sensitivity analysis and present this information also to the designer to help improve performance.

*5.2 Verification of Motion to a Goal Point G with Obstacles*

For our approach to be useful, the environment model needs to be able to accurately represent objects and obstacles to the robot. Figure 10 shows result from the verification of the controller in (3) with an environment model that includes a wall between the initial position P0 and goal G. (The position distribution is shown from an overhead view, as if it were a 2D distribution of points – *it's not of course*, it's a parametric distribution.) Verification in this case returns that the liveness condition does *not* hold, since the controller uses no sensing and cannot detect the wall. The final 2D position distribution Figure 10(c) returned from VIPARS with failure of verification condition shows that the robot position distribution trapped alongside the wall.

In the case of a corner obstacle (Figures 11 and 12) rather than a wall, then it is possible in some conditions for the robot controller of (3) to succeed even without sensing. Uncertainty in motion, as the robot moves towards the goal, may sometimes result in the robot missing the corner (and sometimes in it hitting the corner of course). It's important that VIPARS correctly represent this kind of environment interaction, and to present results to a designer that accurately convey the situation.

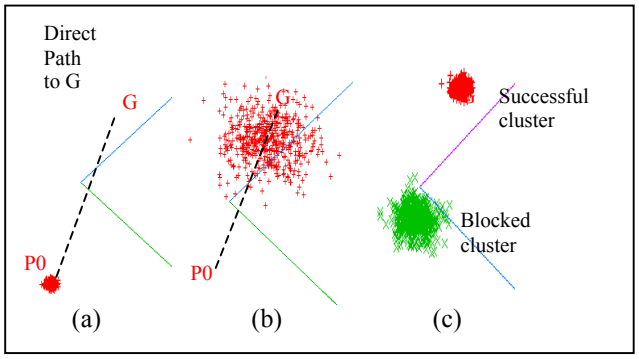


**Figure 11:** Corner obstacle; initial conditions (a) unimodal result (b), multimodal result (b)

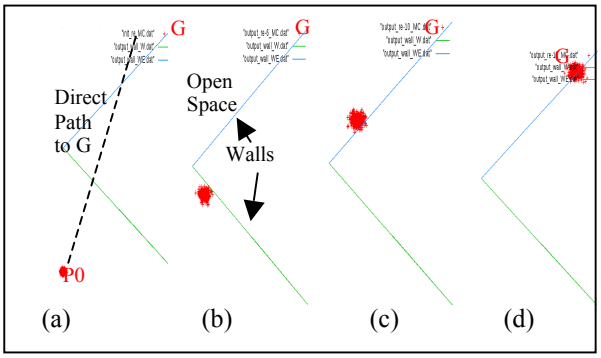


**Figure 12:** Corner obstacle using sensors; Initial conditions (a) close to wall (b,c), final position (d)

Figure 11(a) shows the initial conditions in the case that the path to G is blocked. Figure 11(b) shows the position distribution resulting from a successful verification of the liveness condition. Note however that the final position distribution has expanded (a large variance) so that it covers both the positions in which the robot hits the wall *and also* the positions that reach G. This combination of the two possible result positions is an effect of restricting position to be a normal (single Gaussian) distribution. It is clearly *not* intuitive; we can illustrate the effect of having used a Mixture of Gaussians (MoG) model by tailoring the program to represent the stuck, and the successful cases as separate distributions and these are shown overlaid in Figure 11(c). Comparing Figure 11(b) and 11(c) which represent the same situation: Figure 11(c) breaks the two clusters of spatial positions out, which may be more useful for a designer to see, but demonstrates that a multimodal representation is needed to accurately convey situations like this, where the distribution of robot location has two or more separate clusters. We are currently implementing a MoG version of VIPARS.

Robot programs rely on sensor information to address uncertainty about obstacle locations. The results shown in Figure 12 are a verification of a controller program which uses a sensor to detect blockage by the wall, operating in this same environment with a corner obstruction. The effect of the use of sensors in the code is evident by the location of the position distribution in the intermediate snapshots Figure 12(b,c). The final position distribution doesn't show the variance expansion effect of Figure 11(b) since the controller's use of sensors prevents collisions with the wall.

*5.3 Validation of Motion to a Goal Point G*

The results in the previous two subsections demonstrate that VIPARS verifications produce spatial distributions that are qualitatively consistent with expected robot behaviors. To demonstrate quantitatively consistent results we first carried out measurements of actual robot positions moving at various velocities and distances on an indoor, lab-floor

**Table 3:** $\chi^2$-test data set B for three velocities for a 10m traverse.

| Vm/s | X m | μ m | $\chi^2$-crit | $\chi^2$ |
|---|---|---|---|---|
| 0.2 | (9.745,0.141) | (9.745,0.155) | 6.060 | 0.2579 |
| 0.5 | (9.751,0.123) | (9.769,0.136) | 6.059 | 0.8790 |
| 0.8 | (9.783,0.149) | (9.761,0.161) | 6.06 | 1.352 |

**Table 4:** $\chi^2$-test data set B for three velocities for a 14m traverse.

| Vm/s | X m | μ m | $\chi^2$-crit | $\chi^2$ |
|---|---|---|---|---|
| 0.2 | (13.64,0.217) | (13.65,0.256) | 6.059 | 0.1087 |
| 0.5 | (13.65,0.172) | (13.67,0.227) | 6.057 | 0.7440 |
| 0.8 | (13.67,0.225) | (13.68,0.241) | 6.059 | 0.3531 |

surface: velocities 0.2 m/s, 0.3 m/s and 0.5m/s and distances 2m, 5m, 7m and 10m. These measurements, data set A, were used to calibrate the velocity-dependent motion uncertainty distribution $\Theta$. A series of verification experiments were then carried out in VIPARS for these velocities and distances, and a separate set of actual robot measurements collected, data set B, at 10m and at 14m, one set for each VIPAR experiment. The population spatial distributions generated by VIPARS were compared to data set B using a $\chi^2$-test to test the validation sample mean (X) against the VIPARS predicated mean ($\mu$) and variance ($\Sigma$). The results for the 10m and 14m validation sets are shown in Table3 and Table 4 (with $\Sigma$ omitted for space). These results strongly support the null hypothesis that $X=\mu$ given a minimum error of *5mm*, showing that VIPARS has the potential to produce accurate behavior predictions of the real robot.

## 6. Conclusions and Future Work

This paper has presented results from a novel formal framework for the analysis of probabilistic and behaviour-based robotics algorithms for applications in particular in search, analysis and recovery tasks such as those in the domain of counter-weapons of mass destruction. VIPARS is an algorithm for verifying whether a combination of robot program and robot, sensor, and environment models will satisfy a performance criterion. It exploits a unique state regularity of the behavior-based robotics domain, the system period, to verify probabilistic robotic behavior in terms of probability distributions. The results presented in this paper demonstrate robot program verification for situations including robot and sensor uncertainties (using empirical measurements from a Pioneer 3-AT robot), interaction with objects in the environment, and the use of sensors to guide behavior.

These results concentrated on spatial distributions because it is a crucial point of novelty, not offered by other verification systems to our knowledge, for verification of robot software. Our initial distributions were all single Gaussian distributions and hence unimodal, but as is clear from Figure 13, multimodal distributions are a better match. We are currently implementing a Mixture of Gaussians representation, as well as support for the inverse exponential distribution and combinations such as severities [19]. The validation comparisons of VIPARS-predicted and actual motion behavior for the Pioneer 3-AT robot moving under uncertainty reported in this paper show that VIPARS can be significantly accurate in its predictions. Future work will include more extensive validation of performance predictions including the use of the NIST robot benchmarking testbed [14].


## ACKNOWLEDGEMENT

This research is supported by the Defence Threat Reduction Agency, Basic Research Award #HDTRA1-11-1-0038.


## REFERENCES


[1] Lyons, D., Arkin, R., Nirmal, P and Jiang, S., Designing Autonomous Robot Missions with Performance Guarantees. *IEEE/RSJ IROS, Vilamoura Portugal,* Oct. 2012.

[2] Lyons, D., Arkin, R., Fox, S., Jiang, S., Nirmal, P., and Zafar, M., "Characterizing Performance Guarantees for Real-Time Multiagent Systems Operating in Noisy and Uncertain Environments", *Proc. Perf. Metrics for Int. Sys. (PerMIS'12)*, Baltimore MD, 2012.

[3] MacKenzie, D., Arkin, R.C., Cameron, R., Multiagent Mission Specification and Execution. *Aut. Robots*, 4(1) 1997, pp. 29-52.

[4] Arkin, R.C., *Behavior-Based Robotics*, MIT Press, Cambridge, MA, 1998.

[5] Thrun, S., Burgard, W., and Fox, D., *Probabilistic Robotics*, MIT Press. 2005.

[6] 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).

[7] Arkin, R.C., Lyons, D.M., Jiang, S., Nirmal, P., Zafar, M., Getting it Right the First Time: Predicted Performance Guarantees from the Analysis of Emergent Behavior in Autonomous and Semi-autonomous Systems, *SPIE Unmanned Sys. Technology XIV*, Baltimore MD, 2012.

[8] Endo, Y., MacKenzie, D., and Arkin, R.C., Usability Evaluation of High-level User Assistance for Robot Mission Specification. *IEEE Trans. SMC* Vol. 34, No. 2, pp.168-180, May 2004.

[9] MacKenzie, D., Arkin, R., Evaluating the Usability of Robot Programming Toolsets. *Int. J. Rob. Res.*, Vol. 4, No. 7, April 1998, pp. 381-401.

[10] Yu, W. et al. Analysis and Experimental Verification for Dynamic Modeling of A Skid-Steered Wheeled Vehicle. *IEEE Trans. on Rob., 26(2) Apr. 2010.*

[11] Ben-Ari, M., Principles of the SPIN Model Checker, Springer-verlag: London 2008.

[12] Kwiatkowska, M., Norman, G., and Parker, D., PRISM 4.0: Verification of Probabilistic Real-time Systems. *Proc. 23rd Int. Conf. Computer Aided Verification*, v6806 LNCS, Springer, 2011.

[13] Jhala, R., Majumdar, R., Software Model Checking. *ACM Computing Surveys,* V41 N4, Oct 2009.

[14] Jacoff, A., Messina, E., Huang, H., Virts, A., Downs, A., and Norcross, A., Standard Test Methods for Response Robots ASTM, Aug. 2011.

[15] Baeten, J., A Brief History of Process Algebra. *Elsevier J. Theoretical Comp. Sci. – Process Algebra*, 335(2-3), 23 May 2005

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

[17] Boem, C. and Jacopini, G., "Flow diagrams, Turing machines and languages with only two formation rules, CACM 9(5) 1966.

[18] Arkin, R.C., Motor Schema-Based Mobile Robot Navigation. *Int. Journal of Robotics Research*, Vol. 8, No. 4, Aug. 1989, pp. 92-112.

[19] Yang, D.W., Nadarajah, S., Drought modeling and products of random variables with exponential kernals *Stoch. Env. Res. & Risk Ass. V21 N2 2006.*