Performance Verification for Behavior-based Robot Missions[*]

D. Lyons[1], R. Arkin[2], S. Jiang[2], T-L. Liu[1], P. Nirmal[1], J. Deeb[2]

[1]Fordham University, Bronx NY, USA
[2]Georgia Institute of Technology, Atlanta GA, USA

Abstract. Certain robot missions need to perform predictably in a physical environment that may only be poorly characterized in advance. This requirement raises many issues for existing approaches to software verification. An approach based on behavior-based controllers in a process-algebra framework is proposed by Lyons et al [15] to side-step state combinatorics. In this paper we show that this approach can be used to generate a Dynamic Bayesian Network for the problem, and that verification is reduced to a filtering problem for this network. We present validation results for the verification of a multiple waypoint robot mission using this approach.

1 Introduction

In research being conducted for the Defense Threat Reduction Agency (DTRA), we are concerned with missions that may only have a single opportunity for successful completion with serious consequences if the mission is not completed properly. In particular we are investigating missions for Counter-Weapons of Mass Destruction (C-WMD) operations, which require discovery of a WMD within a structure and then either neutralizing it or reporting its location and existence to the command authority. Typical scenarios consist of situations where the environment may be poorly characterized in advance in terms of spatial layout, and often have time-critical performance requirements. It is our goal to provide reliable performance guarantees for whether or not the mission as specified may be successfully completed under these circumstances, and towards that end we have developed a set of specialized software tools to provide guidance to an operator/commander prior to deployment.

In our research, behavior-based robot missions are designed and carried out using the Georgia Tech *MissionLab* [17- 19] toolkit. The mission designer specifies:

1. A performance guarantee that she wishes to verify the mission against (e.g., that the robot traverses a set of waypoints within a certain time to a given probability of success, or that it locates a specific target within a certain time to a certain probability of success); and,
2. A partial description of the environment in which the mission will be carried out.

In prior work [2,13-16] we have introduced a process algebra approach to the representation and analysis of robot programs and robot environments. In [15] we intro-

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

duced an algorithm to extract periodic behavior, in the form of a set of recurrent flow functions, from a set of concurrent, connected processes that represent a behavior-based robot program and its environment. The effect of motion and sensor uncertainty is crucial in real-world robotics applications. In this paper, we address the problem of how flow-functions that include random variables can be extracted from process networks. We show that a flow function can be mapped to an equivalent Bayesian Network, and that the problem of determining whether a mission will achieve its performance guarantee can be reduced to the filtering problem for a Dynamic Bayesian Network. In prior work [16] we have statistically validated our results using a simple, single move robot mission. Here we present new results for a multi-waypoint robot mission and we validate these results to show the predictive power of our method.

2 Prior Work

Automatic verification of software is a very desirable functionality in any application where software failure can bring heavy penalties [7]. Examples include embedded software such as airplane and space flight controllers as well as factory controllers and medical equipment. While we know that a completely general solution is ruled out by the undecidability of the halting problem, much research has been conducted on restricted instances of the problem. Model checking has been a very successful technique [6, 8] where a program is cast as a state-based transition system in which states are labeled with sets of propositions, a *Kripke* system. The instructions in the program map from one state to a successor state. If the program has *n* variables, and if each variable r_i can have values from a set $val(r_i)$, then the state space of the program is $\Pi_i \; val(r_i) = val(r_0)\times \ldots \times val(r_{n-1})$. The verification problem in model-checking is, at its heart, a test of the reachability of a state or set of states from the start state given the program instructions. The combinatorics involved in $\Pi_i \; val(r_i)$ have always been clear, and model-checking approaches are typically divided into enumerative methods that search this (perhaps huge) graph of states, and symbolic methods which instead explore (a smaller number of) sets of these states [8].

Automated verification of robot and multirobot software has several characteristics that set it apart from general purpose software verification. The first is that the robot program does not execute based on static inputs, but rather interacts with an environment in an ongoing fashion. This is recognized in discrete-event control by considering the system as a concurrent composition of the robot program and an environment (plant) model [21]. From a model-checking perspective, the system states-space is now increased beyond the program state-space by the product of environment variables. A second characteristic is that there may be a necessary continuous nature to some aspects of the environment. Since a state transition approach is necessarily discrete, various hybrid continuous-discrete systems [12] have been introduced to handle this. Finally, significant uncertainty pertains to the result of robot sensing and motion; this cannot be ignored or the results are not realistic.

While there have been some successful applications of model-checking to discretized versions of the robot programming problem [11], we need to adopt an approach that produces verification results that will be validated when tested experimentally with C-WMD robot missions. We expect that a state-based approach will experience significant combinatorial problems due to the characteristics discussed in the previous paragraph. So rather than a hybrid state/continuous approach, we have opted to avoid discussing state at all costs.

In [15] we introduce a process algebra (PA) approach to representing robot programs and environment models. Karaman et al. [10] also use a PA as a specification language for multiple UAV missions and develop a polynomial time algorithm that produces a plan to satisfy the specification. That work, and our previous work in PA for performance analysis of robot programs [13], leveraged the trace, or history of events, of a process. In this work, we use a PA that includes I/O port communications [23]. The advantage of the PA is that it can be used to determine how a process transforms its inputs to produce outputs without reference to states.

We leverage this approach in [15] for a specific kind of robot programming: behavior-based robot programming [1]. A behavior-based robot interacting with its environment will respond to a specific set of environmental percepts as programmed by its behaviors. Once a precept is responded to, the robot may return to this behavioral state or move to another that handles a different set of precepts. For the specific case of a system of environment model and programmed behaviors represented as *tail-recursive* process definitions, we proposed a novel interleaving theorem [15] that allowed us to identify a single composite *system period process*. This period process contained all the port to port communications that could happen in the system as part of the precept-response cycle. In a subsequent step, we showed how the transformations that occur with these port communications can then be written as a set of recurrent functions which we called flow-functions since they related the value of variables in one iteration of the system period to the value in the next iteration. In this way, we separated the issue of variable values and variable transformations from the concept, and limitations, of state. The verification problem in this framework is the solution of this set of recurrent flow functions, and in [15] we proposed to use a package such as PURRS (Univ. of Palma recurrent equation solver) for this.

Uncertainty plays a major role in real-life robotic performance and needs to be included in any useful approach to robot verification. Napp and Klavins [20] introduce a guarded command language CCL for programming and reasoning about robot programs and environments. They address uncertainty by adding a concept of rates and exponential probability distributions to CCL, which allows them to reason about the robustness of programs. Johnson and Kress-Gazit [9] develop a model-checking algorithm that handled uncertainty based on Discrete Time Markov Chains; however, they comment on the intractability of their approach for large state spaces.

With the objective of continuing to avoid a state-based approach, we propose here to augment flow-functions to include random variables, and to map the solution of a system of flow-functions to a filtering problem for a Dynamic Bayesian Network. This approach would continue to bypass state combinatorics and would allow the use of various parametric uncertainty distributions including Mixture of Gaussians [22] to capture motion and sensor uncertainty.

3 Mission Specification

Dull, dirty, and dangerous missions are considered to be the natural niche for robots, and these missions have been a major driving force behind the advancement of robot technology. Over the past decades, we have seen an increasing number of robots being deployed to accomplish dangerous missions (e.g., disarming IEDs in Afganistan). Missions in the domains of urban search and rescue (USAR) and counter weapons of mass destruction (C-WMD) are not only dangerous, but their failures usually have dire consequences. It is highly desriable then to have the ability to verify the performance of a robot before it is deployed to carry out a mission. However, verification of robotic missions poses a unique and great challenge that is is different from traditional software verification – the robot has to work in the real world, and the real world is inherently unpredictable. For example, robots were deployed during the World Trade Center rescue response, where the environment had become highly unstructured and filled with rubble piles [5]. In this paper, we present our ongoing work [2,15] on a verification framework for performance guarantees for critical missions where failure is not an option – the robot has to get it right the first time.

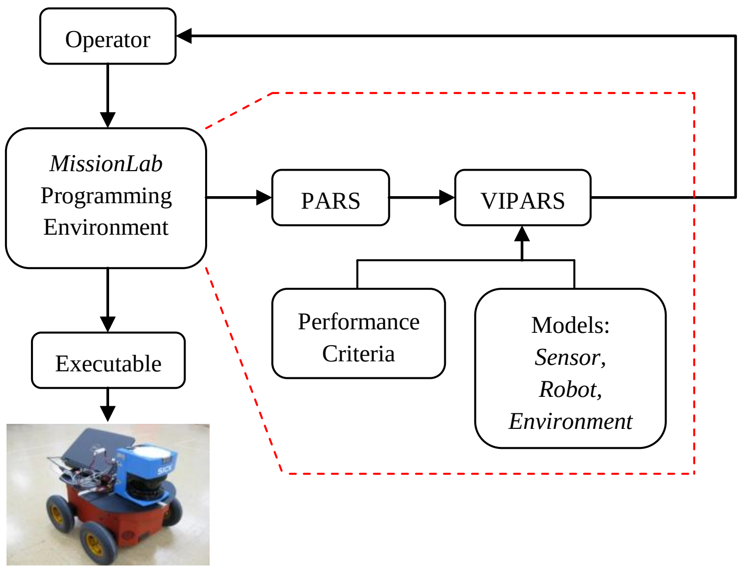

Operator
MissionLab Programming Environment
PARS
VIPARS
Performance Criteria
Models: Sensor, Robot, Environment
Executable

Fig. 1. *MissionLab* robot mission specification toolset with VIPARS verification module

We have built our robot mission verification framework upon *MissionLab*, a behavior-based robot programming environment [18]. *MissionLab* provides a graphical user interface where robot programs can be constructed as a finite state automaton (FSA) from a library of primitive behaviors. One of the many unique features of *MissionLab* is that it generates hardware-independent executables from user-constructed FSAs, which allows the desired robot platform to be chosen at run time. For critical missions where performance guarantees are desirable, we introduced a verification framework into *MissionLab* where the missions can be verified before the executable generation step.

The verification framework is shown in Fig. 1 as an extension to the *MissionLab* programming environment. The core of the framework is the process algebra based verification module, Verification in Process Algebra Robot Schemas (VIPARS) [15]. To initiate the verification of a mission, the robot program is compiled to PARS (Process Algebra Robot Schemas), the language understood by VIPARS. The robot operator also needs to provide VIPARS with models of the robot, the sensors it is equiped with, and the environment it is to operate in, along with the performance criteria that the mission is required to meet. VIPARS provides the operator with the performance guarantee for the mission based on how well the provided performance criteria were met. The verification module effectively forms a feedback design loop, where the operator can iteratively refine the robot program based on the information provided by VIPARS (such as e.g., time criterion is not met).

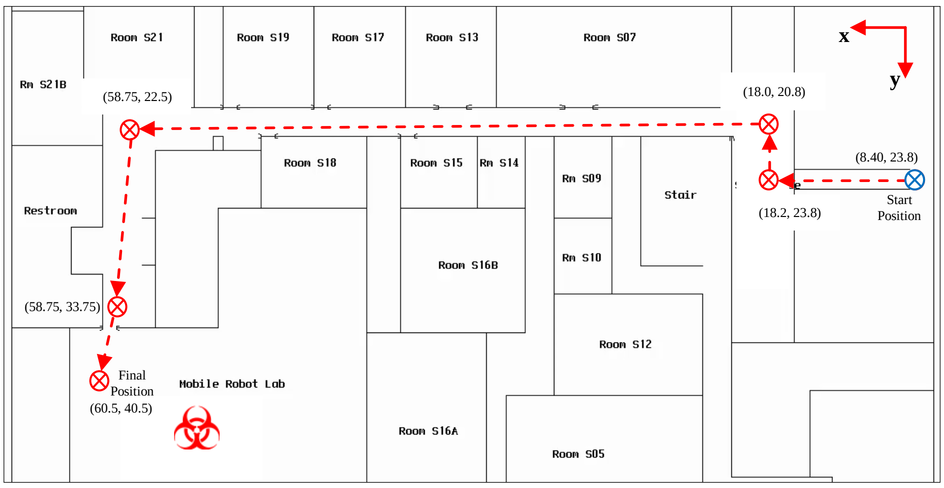


Fig. 2. Building layout with waypoints labeled in red

3.1 Mission Design

To illustrate the process of designing a mission with *MissionLab* and verifying it with VIPARS, we present a biohazard search scenario where the robot needs to access a room inside the basement of a building, where potential biological weapons might be located. The layout of the basement is shown in Fig. 2, and the room the robot needs to access is shown with a red biohazard symbol. With a known layout of the environment, the simplest solution to accomplish the mission is to designate waypoints which the robot can follow to access the room of potential threat. The waypoints and the path of travel are shown in red in Fig. 2. However, it is more often than not that we do not have such a strong knowledge of the operating environment. In these cases, the simple waypoints-based solution would not be appropriate. Autonomous exploration missions without such strong knowledge of the environment will be addressed in later work. Nonetheless, while the waypoint-based mission is simple to design, it still provides substantial challenges for verification due to uncertainties in robot localization and obstacle detection.

The design of the FSA for the multi-waypoints mission from Fig. 2 is shown in Fig. 3, which was created with *CfgEdit*, the Configuration Editor, in *MissionLab*. The FSA consists of a series of *GoToGuarded* and *Spin* behaviors with *AtGoal* and *HasTurned* triggers. The *GoToGuarded* behavior drives the robot to a specified goal location (i.e., waypoint) with a guarded radius of velocity dropoff around the goal location. The *AtGoal* trigger causes a transistion to the next state when the robot reaches the goal location. The *spin* behavior rotates the robot around an obstacle with a given velocity. The *HasTurned* behavior causes a state transition when the robot has turned a desired angle. The robot operator could verify the design intent by simulating the mission with the simulation environment provided in *MissionLab*, however this is not sufficient to provide performance guarantees for the mission.

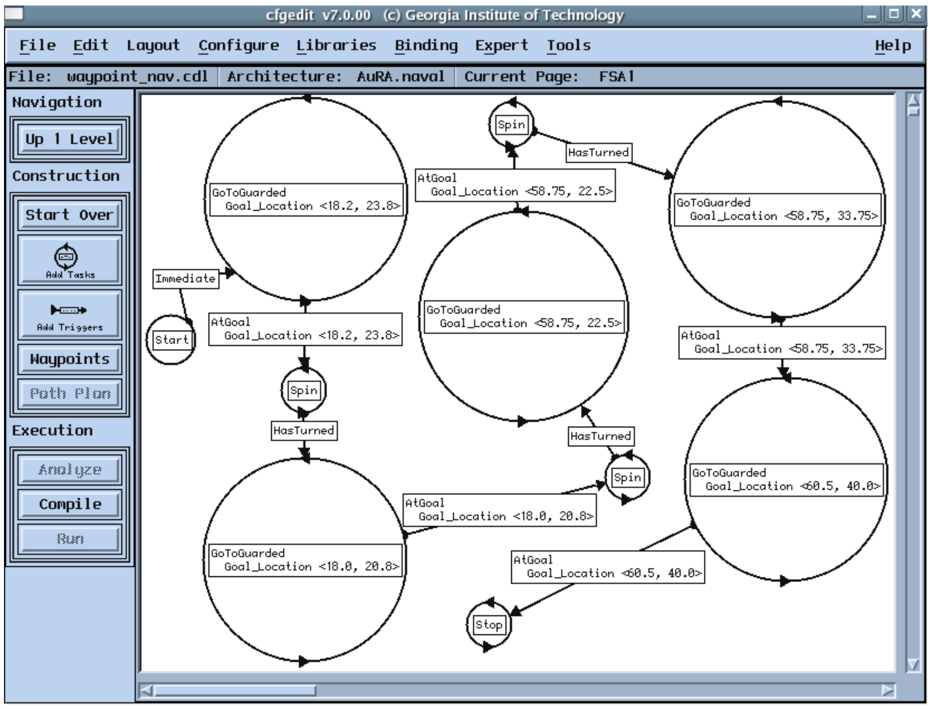

cfgedit v7.0.00 (c) Georgia Institute of Technology
File Edit Layout Configure Libraries Binding Expert Tools Help
File: waypoint_nav.cdl Architecture: AuRA.naval Current Page: FSA1
Navigation
Up 1 Level
Construction
Start Over
Add Tasks
Add Triggers
Waypoints
Path Plan
Execution
Analyze
Compile
Run
Start
Immediate
GoToGuarded Goal_Location <18.2, 23.8>
AtGoal Goal_Location <18.2, 23.8>
Spin
HasTurned
GoToGuarded Goal_Location <18.0, 20.8>
AtGoal Goal_Location <18.0, 20.8>
Spin
HasTurned
GoToGuarded Goal_Location <58.75, 22.5>
AtGoal Goal_Location <58.75, 22.5>
Spin
HasTurned
GoToGuarded Goal_Location <58.75, 33.75>
AtGoal Goal_Location <58.75, 33.75>
GoToGuarded Goal_Location <60.5, 40.0>
AtGoal Goal_Location <60.5, 40.0>
Stop

Fig. 3. Mission design with *MissionLab*'s *CfgEdit*

3.2 Verification of Performance Guarantee

"I have not failed. I've just found 10,000 ways that won't work." – Thomas Edison

Designs rarely work coming off the drawing board the first time. Final working products usually emerge only through numerous "going back to the drawing board" moments. The design of robot missions is no exception. However, for time-critical C-WMD and USAR missions where we might only have one opportunity to attempt the mission, we need to have some guarantee that our robotic system will succeed before its deployment. It is the objective of our onging work to provide robot mission designers/operators with such performance guarantees.

To obtain a performance guarantee for the robot FSA in Fig. 3, the operator needs to compile the robot program into PARS and provide VIPARS with the performance criteria and models of sensor, robot, and the environment (Fig. 1). Performance criteria are mission constraints (e.g., safety and time constraints) that the robot system has to meet in order to assert "mission accomplished." The robot we used for the multi-waypoints mission is a four-wheeled skid-steered mobile robot, the Pioneer 3-AT, as shown in Fig. 4. The robot is equiped with wheel encoders for localization, a gyro for heading correction, and a SICK laser for obstacle avoidance. The sensor, robot, and environment models are provided as libraries where the operator would select the necessary models to be used for verification [2].

Currently, VIPARS outputs 1) a Boolean answer to whether the mission can be successful, and 2) a set of variables in the performance criteria indicating if each criterion had been met. If the predicted performance of the mission does not meet the necessary performance criteria, the operator could refine the robot program based on the feedback provided by VIPARS. This iterative process can continue until the operator is satisfied with the performance guarantee and sufficiently confident to deploy the robot.

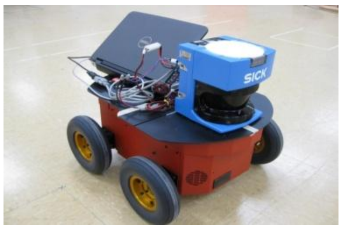


Fig. 4. Pioneer 3-AT

4 PARS Representation of Missions

PARS is a process algebra [3] for representing robot programs and environments for the purpose of analysis and verification. This section gives a brief introduction to PARS as a precursor to the discussion on flow-functions and filtering in subsequent sections and finally to the presentation of the validation results. For a more thorough introduction, and wider selection of controllers and environment models, see [13-16].

The semantics of a process in PARS is an extended port-automaton [23], 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 $\mathbf{P}$ 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:

$$\mathbf{P}_{u1,u2,\dots}\,(i1,i2,\dots)\,(o1,o2,\dots)\,\langle v1,v2,\dots\rangle \tag{1}$$

For brevity, the parts of a process description that are empty are typically omitted, and in this paper (except for Section 6, which refers to actual implementation) we will also use global port names rather than the more general but more wordy port connections. Process variables (initial parameters, results) can be random variables; we'll come back to this in more detail. 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 $\mathbf{T}$ that inputs a value on port *c1* and then outputs it on port *c2* is defined as a sequential composition (;) as follows:

$$\mathbf{T} = \mathbf{In}_{c1}\langle x\rangle\,;\,\mathbf{Out}_{c2,x} \tag{2}$$

A sequential composition in which the first process ends in abort (see Table 1) just aborts; this implements a conditional construct. Other composition operations include parallel-max (|) and parallel-min or disabling (#). A tail-recursive (TR) process is written as:

$$\mathbf{T}_a = \mathbf{P}_a\langle b\rangle\,;\,\mathbf{T}_{f(a,b)} \tag{3}$$

This provides an iterative construct. Any language that implements sequence, condition and loop constructs is sufficient to represent any program [4]; thus, we can be confident that PARS can represent any program. In (3), *f(a,b)* indicates how the parameters (or variables) of the process are transformed when passed to the next recursion. We refer to such functions for TR processes as *parameter flow functions*.

Table 1. Examples of Basic Processes

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

4.1 PARS Controllers

One objective of our project is to automatically translate *MissionLab*'s underlying CNL mission specification language [19] into the PARS description of the mission controller. This work is in progress but not completed, and for now, we manually translate from CNL to PARS. A *MissionLab* waypoint mission, as described in Section 3, might be approximated in PARS as:

$$\begin{aligned}
\mathbf{Mission}_{w,i} &= \mathbf{Goto}_{w(i)}\,;\,\mathbf{Neq}_{i,n}\,;\,\mathbf{Mission}_{w,i+1} \\
\mathbf{Goto}_{a} &= \mathbf{TurnTo}_{a}\,;\,\mathbf{MoveTo}_{a} \\
\mathbf{MoveTo}_{g} &= \mathbf{In}_{p}\langle r\rangle\,;\,\mathbf{Neq}_{r,g}\,;\,\mathbf{Out}_{v,\,u(g-r)}\,;\,\mathbf{MoveTo}_{g} \\
\mathbf{TurnTo}_{g} &= \mathbf{In}_{p}\langle r\rangle\,;\,\mathbf{Out}_{h,\,d(g-r)}
\end{aligned} \tag{4}$$

The controller $\mathbf{Mission}_{w,0}$ visits a series of waypoints *w(i), i=0..n*. For each waypoint, $\mathbf{Goto}_{w(i)}$ first turns the robot towards the waypoint by outputting *d(g-r)*, the relative direction to the waypoint, onto the heading port *h*, and then repeatedly outputting a speed, *u(g-r)* on the velocity port *v*.

4.2 PARS Environments

An environment model in PARS is a causal model of the environment in which a robot program is carried out. An example of an environment model that includes both position and heading uncertainty is shown below:

$$\begin{aligned}
\mathbf{Env}_{r,a,s} &= (\,\mathbf{Delay}_{t}\ \#\ \mathbf{Odo}_{r}\,\#\ \mathbf{At}_{r}\,)\,; \\
&\quad ((\,\mathbf{In}_{h}\langle a\rangle\,;\,\mathbf{Ran}_{\Theta h}\langle z\rangle\,)\ \#\ (\,\mathbf{In}_{v}\langle s\rangle\,;\,\mathbf{Ran}_{\Theta v}\langle w\rangle\,))\,; \\
&\qquad \mathbf{Env}_{r+u(a+z)*(s+w)*t,\ a,\,s} \\
\mathbf{Odo}_{r} &= \mathbf{Ran}_{\Phi}\langle e\rangle\ ;\,\mathbf{Out}_{p,\,r+e}\ ;\ \mathbf{Odo}_{r}
\end{aligned} \tag{5}$$

The environment model accepts a heading input on port *h* or a speed in the direction of the heading on port *v*. The process $\mathbf{At}_r$ represents the robot at location *r*. The process **Odo** (short for Odometry sensor) makes position information (with noise) available in a loop until terminated by the **Delay**. The new position of the robot is calculated as the old position incremented by a noisy speed command *(s+w)* in the unit vector direction *u(a+z)* of the noisy heading. The actuator and odometer noise (the variables *z, w,* and *e* in (5)) is characterized by the distributions for speed, heading and sensor noise, $\Theta h \sim N(\mu_h,\sigma_h)$, $\Theta v \sim N(\mu_v,\sigma_v)$, and $\Phi \sim N(\mu_m,\sigma_m)$.

4.3 PARS Goals

It is very common in model-checking and other kinds of verification to use a temporal logic to specify the property to be verified [6]. Making this connection involves constructing a Kripke model so that propositions can be related to model state. We resist identifying states for all the reasons stated in Section 2, and hence we do not adopt a temporal logic model for specifying properties to be verified. Instead we specify goals directly in PARS. For example, the designer may wish to specify that the robot arrives at position *a* after time *t1*:

$$\mathbf{Goal} = \mathbf{Delay}_{t1} \,;\, (\mathbf{Delay}_{t2} \,\#\, \mathbf{At}_{a}) \tag{6}$$

where *t1* and *t2* are variables here not constants. A property specification process network differs from a process network in that it is actually a process network constraint expression, a specification of a set of possible networks. The system and property to be verified are compared and if the system can be shown equivalent to the property, we extract the constraints that the property network impose on the system and determine if they hold. This latter problem is one of the main topics of this paper, and we address this issue in more depth in the next section.

5 Verification Method

The verification approach presented at the end of the last section is a very challenging one. Given a parallel composition of a controller and system:

$$(\,\mathbf{Goto}_{a} \mid \mathbf{Env}_{r0,h0,0}\,) \tag{7}$$

can we verify that this will achieve the property specification in (6)? In prior work [15] we leverage a property of behavior-based systems to reduce the complexity of this problem. This material is reviewed in section 5.1. In short, it matches the recurrent structure in the controller and environment to generate a process network that is a behavioral system period. The port connectivity in this system period is then analyzed to determine the way in which the system period transforms process variables, generating a set of recurrent functions, which we call flow-functions, one f_i for each variable r_i in the system period. We show that verification then consists of solving these recurrent functions for initial variable values and goal variable values (established by matching the system period and property network) as boundary conditions. In this paper we now consider a practical Bayesian approach to the solution of these flow functions.

5.1 SysGen Interleaving Theorem

A behavior-based robot interacting with its environment [1] will respond to a specific set of environmental percepts as programmed by its behaviors. 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 in the same repetitive way. Consider a set of TR process equations **P1**, **P2**, ..., **Pm** that form a system **Sys** through concurrent composition:

$$\mathbf{Sys} = \mathbf{P1} \mid \mathbf{P2} \mid \ldots \mid \mathbf{Pm} \tag{8}$$

$$= \mathrm{S}(\mathbf{P1}, \ldots, \mathbf{Pm}) \ ; \ \mathbf{Sys}$$

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 as shown in (8)? An *interleaving theorem* in process algebra relates sequential and parallel operations and (8) is an example of such. In [15,16] we describe the *SysGen* algorithm that determines a system period $\mathbf{S}(\mathbf{P1}, \ldots, \mathbf{Pm})$, if one can be can be found. We repeat some of this material below, for background for the remainder of the paper.

SysGen [15] starts with the non recursive body $\widehat{\mathbf{Pi}}$ of each TR process, $\mathbf{Pi} = \widehat{\mathbf{Pi}}; \mathbf{Pi}.$ For the purpose of matching input and output operations, each body is then projected to just the port communication processes:

$$\mathbf{IOi} = \widehat{\mathbf{Pi}} \downarrow \{\mathbf{In}, \mathbf{Out}\}. \tag{9}$$

The port to port communication map *cm* specifies how ports on one process connect to ports on another in the network. We restrict *cm* to connect no more than two process together at any time (i.e. no fan-out or fan-in of connections) This information is specified in (1) by the port connections, but in the examples in previous sections, we just gave connected ports the same name, for brevity. Let's call $\mathbf{IOi}(j)$ the *jth* port operation in $\mathbf{IOi}$ and let $p(\mathbf{IOi}(j))$ be portname in that operation. Then SysGen starts with $\mathbf{IOi}(j_i = 0)$ for each process Pi and then checks for:

$$cm(\ p\big(\mathbf{IOi}(j_i)\big),\ p(\mathbf{IOk}(j_k))\ ) \tag{10}$$

For sequential and disabling compositions in $\mathbf{IOi}$, as soon as (10) is identified, j_i and j_k can be incremented to the next operation. For parallel composion, all the operations in the composition need to be matched before j_i is incremented.

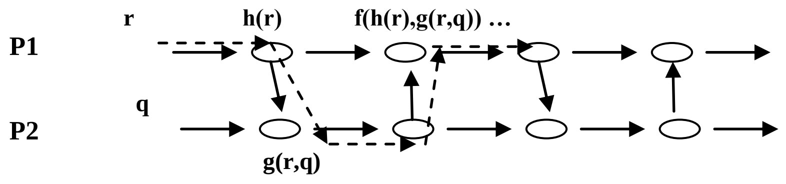


Fig. 5.: Example of variable value transformation (dotted lines) for variables *r* and *q* in a single system period composed of two processes P1 and P2.

If at any point, it is not possible to meet (10), but one or more processes have matched all operations, then those processes can be *unwound*. For example, if $\mathbf{IOi}$ has been matched already, then we can replace it with $\mathbf{IOi} = \mathbf{IOi}\,; \mathbf{IOi}$, and in this way continue matching ports with (10). When all $\mathbf{IOi}$ are completely matched (including any unwound processes), the system period is established; but if (10) fails at any point, then no system period exists. The system period will be:

$$\mathbf{S}(\mathbf{P1}, \ldots, \mathbf{Pm}) = \widehat{\mathbf{P1}}^{k1} |\ \widehat{\mathbf{P2}}^{k2} | \ldots | \widehat{\mathbf{Pm}}^{km} \tag{11}$$

For some constants *k1,…,km*, and where $\mathbf{P}^k = \mathbf{P}^{k-1}$; $\mathbf{P}$ and $\mathbf{P}^0 = \mathbf{P}$. This result allows us to write a flow function for the system **Sys** in terms of the flow functions of the components processes, taking into account the interprocess communiations identified by (10). Thus, we *just* inspect the communications that transpire in *one* system period rather than all communications that could occur during execution; a big reduction in complexity for verification.

5.2 Flow Functions

SysGen allows us to recast the analysis of the recurrent system into the analysis of a single period. This period transforms the values of the variables at start of repetition *k* of the period to those at the start of repetition *k+1*. Variables may be transformed by operations within processes, we can get this information from the process flow functions, or they may be sent via port communications to be included in other processes, but that we now have to calculate. Figure 5 shows an example of this calculation for two processes. The *FloGen* algorithm (Figure 6) produces a flow function that includes these transformations for each flow variable (parameter) of the system period. For each flow variable, $r_i \in R = \{r_1,\dots,r_n\}$, FloGen traces its transformation through processes and port communications to generate a single flow function f_i defined as:

$$f_i(r_1,\dots,r_n):\ \ \mathrm{val}(r_{1,k})\times..\times\mathrm{val}(r_{n,k})\ \rightarrow \mathrm{val}(r_{i,k+1}) \tag{12}$$

FloGen($FS = \{f_1,\dots f_m\}$): // component flow functions for processes *p1,…,pm*

```
1. For each $f_i \in FS$
2.   For each $v_j$ in $f_i$ not a parameter of $pi$
3.     $a \leftarrow$ port in $pi$ that generated $v_j$
4.     While ($a$ != $\perp$)
5.       $cm(a)$ is the network connection of $a$ on $pk$
6.       $u \leftarrow$ parameter value to the port operation on $cm(a)$
7.       $a \leftarrow$ port in $pk$ that generated $u$ or $\perp$ if none
8.     Replace $v_j$ with $u$
```

Fig. 6. Flow Function Generation Algorithm, *FloGen*

The complexity of FloGen depends on the number of component processes and the number of parameters to each, since each parameter will generate one flow function. If there are *m* port-to-port connections in the system period, then *m* is the upper bound on the sequence of substitutions for port connections in FloGen.

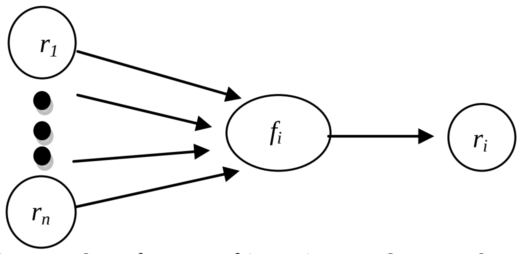


Fig. 7. Flow function $f_i(r_1\dots r_n)=r_i$ evaluation shown as a Bayesian Network

Flow variables may be random variables. Hence the flow function relates the value of the random variable $r_{i,k}$ of time step *k* to its value in time step *k+1* given the values of the other variables in *R*. This is equivalent to a calculation of the posterior probability $r_{i,k+1}$ given the values of all the variable values at time *k*, R_k, which we can write

$$P(\, r_{i,k+1} \mid R_k) \;= f_i(\, R_k). \qquad (13)$$

The result of matching a goal network and a system is a constraint on the posteriori values of some of the flow variables.

Not all variables in R_k may be needed to calculate each r_{k+1}. Any particular variable may only depend on some of the variables in R_k as given by the structure of the processes and process communications. This structural locality property is identified by the FloGen algorithm as it follows port connections between processes (Fig. 5), expressing the inherent conditional independence:

$$P(\, r_{i,k+1} \mid R_{i,k}) \;= f_i(\, R_{i,k}),\, R_{i,k} \subseteq R_k \qquad (14)$$

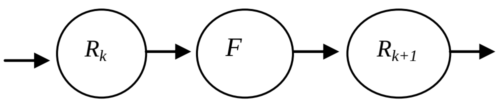


Fig. 8. Dynamic Bayesian Network

The resulting structure can be drawn as a Bayesian network as shown in Figure 7. As long as flow functions can include the effect of program conditionals [16], we can assume $R_{i,k} = R_i$ and hence that the evolution of flow-variable values is a stationary process and can be captured as the Dynamic Bayesian Network (DBN) shown in Figure 8. Those flow variables that are used in the calculation of other flow variables are given by $\bigcup_i R_i$ and are exactly the variables whose flow-fuction transformations compose the transition model for the DBN.

Finally, we define the transition model of the DBN as the function F, where

$$F(R_k) = (\, f_1(R_{1,k}),\, f_2(R_{2,k}), \ldots\,) \qquad (15)$$

5.3 Verification as Filtering

Matching the system and goal networks (Section 4.3) identifies a subset of the flow-variables, $G \subseteq R$, and the values to be associated with them

$$GV = \{\, (g,v) \mid g \in G \text{ and } v \in val(g)\, \} \qquad (16)$$

The verification problem asks whether the execution of the controller in the given environment will result in the the flow-variables in G having the values specified by GV. However, if the variables are random variables, then we need to modify this: $P(GV_k \mid R_k)$ is the probability that GV holds at step k given the flow-variable values at that step. For each $g \in G$ this means integrating the value of the probability density over a small range around the value v. Our definition of a successful verification is:

$$P(GV_k \mid R_{1:k}) > P_v \qquad (17)$$

where P_v is a user specified constant (typically 80% in our experiments, but user definable) and where $R_{1:k}$ means the sequence of flow-variable values from the first step to step k. We introduce an observation model $GF(R_k)$ to implement this evaluation:

$$GF(R_k) \quad = \quad P(GV_k \mid R_k) \tag{18}$$

The goal conditions may be achieved on any step, so the probability of achieving the goal is the disjunction of the probabilities on each step:

$$P(GV_k \mid R_{1:k}) = P(GV_1|R_1) + P(GV_2|R_{1:2}) + \ldots \tag{19}$$

$$+ P(GV_{k-1} \mid R_{1:k-1}).$$

We can write this more compactly as:

$$P(GV_k \mid R_{1:k}) = \sum_{i=1}^{k} P(GV_i \mid R_i) P(R_i \mid R_{1:i-1}) \tag{20}$$

And since we know that each R_i is linked to the one before in the DBN by our transition model $R_{i+1} = F(R_i)$, and our goal satisfaction is related to R_i by the observation model $GF(R_i)$ we can continue:

$$P(GV_k \mid R_{1:k}) = \sum_{i=1}^{k} P(GV_i \mid R_i) \prod_{j=1}^{i} P(R_j \mid R_{j-1}) \tag{21}$$

$$= \sum_{i=1}^{k} P(GV_i \mid R_i) F^i(R_1)$$

$$= \sum_{i=1}^{k} GF(F^i(R_1))$$

While P_v gives a way to determine a successful verification, it does not allow us to determine a non-successful verification. One solution is to bound k, insisting that:

$$P(GV_k \mid R_{1:k}) > P_v \text{ and } k < K_{max} \tag{22}$$

This solution is reasonable for example if k can be related to time (for example if a maximum time can be established for the execution of a system period) and if there is a maximum time constraint on the activity (for example, that the mission must be achieved before a given time has elapsed).

5.4 Extension of SysGen

SysGen is defined only for a concurrent composition of TR processes [15]. It is clear that a straight-line sequence of several processes is not be a TR process! Furthermore this is a process structure we expect to see in behavior-based systems, as we switch from responding to one set of precepts to responding to another fro example. As a more immediate matter, we also see this here when we sequence a series of **Goto**$_a$ processes in a multiple waypoint mission. Luckily, there is a straightforeward extension for SysGen. Consider one process **Pi** in (8) to be non-TR, and let us consider the scenarios:

1. $\mathbf{Pi}$ is pure straight-line code: In that case, we can say $\widehat{\mathbf{Pi}} = \mathbf{Pi}$, calculate its flow-function and DBN, and filter the DBN for just a single time-step (since the straight-line code does not repeat, only one step is necessary).
2. $\mathbf{Pi}$ is straight-line code followed by a single TR process, $\mathbf{Pi}_{a,b} = \mathbf{SL}_a\langle y \rangle \,;\, \mathbf{T}_{a,b,y}$. In this case, we break the problem into two sequential problems;
   (a) we first address the system (8) with $\mathbf{Pi}$ replaced by $\mathbf{SL}$, calculating the flow-function and DBN and filtering for one time step, and then *carry the variable values over to a second system* where
   (b) we address the system (8) with $\mathbf{Pi}$ replaced by $\mathbf{T}$, which is TR and hence can be handled in the normal fashion.
3. $\mathbf{Pi}$ is a sequence of two TR processes, $\mathbf{Pi}_{a,b} = \mathbf{T1}_a\langle y \rangle \,;\, \mathbf{T2}_{a,b,y}$. We also break this into a sequence of two problems with $\mathbf{Pi}$ replaced by $\mathbf{T1}$ in the first and $\mathbf{Pi}$ replaced by $\mathbf{T2}$ in the second, carrying the variable values over between both problems.

This extends the SysGen result to handle the common pattern of any combination of straight-line and TR processes, though it is limited to just one process in the system, and which we will map to the *MissionLab* FSA process.

5.5 Verification Examples

The following are some prior examples of VIPARS results to make the preceding theory more concrete. In each case here, the robot controller attempts to move the robot from a point P0 to a point G, a single waypoint. The condition being verified in (22) is that the robot is at the point *G* after some time $t<T_{max}$ with probability $p>P_{min}$.

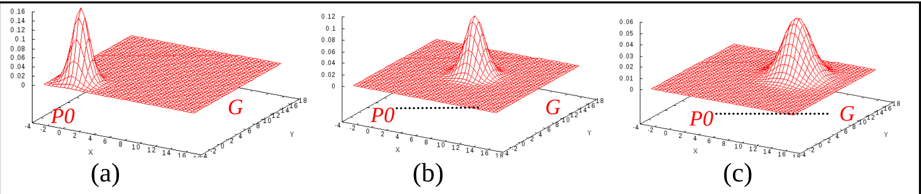


Fig. 9. Three snapshots of the robot position distribution, from P0 (a) to G(c).

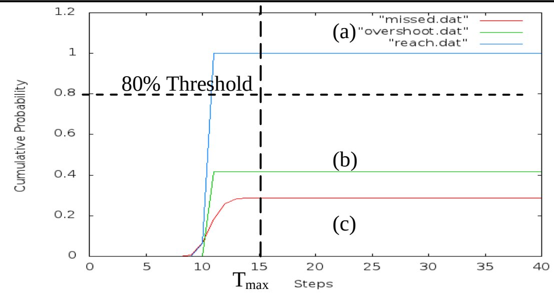


Fig. 10. Cumulative probability of the Goal Condition versus DBN iteration step.

The controller (4) and environment model (5) were used to build a single waypoint mission from an initial position P0 to a goal location G and submitted to VIPARS. Figure 9 shows the value of the position distribution at several steps during verification of that mission. Figure 10 shows the value of the probability of the goal condition as a function of the DBN iteration step. In Figure 10(a), the cumulative probability of being at G rises monotonically as the robot approaches G. The initial low probabilities represent the cases when the robot motion error is so small that the robot arrives at the goal relatively quickly. VIPARS returns the position distribution for this mission at the iteration step where the probability of having arrived at G exceeds the (mission-designer) specified threshold (80% in this example).

If the cumulative probability is not above the threshold P_{min} before T_{max} then the verification returns that this goal condition was not met. A system can fail to meet the performance criterion if the robot controller is logically defective (e.g., Figure 10(b)) of course, but also if the motion and sensing noise is just too large (e.g., Figure 10(c)).

For our approach to be useful, the PARS environment model needs to be able to represent objects and obstacles when they are known. In other work [13, 16], we have shown PARS environment models that include walls, obstacles and different terrain types. Figure 11 below shows a *Mixture of Gaussian* (MoG) VIPARS position distribution result for a waypoint mission through a narrow doorway and corridor. The MoG members are shown as shaded 1SD ellipses, the shading indicating the weight of the member. The smaller clusters to each side of the doorway in Figure 11(b,c) indicate the probability of missing the door and hitting the wall. The member cluster smeared out in the corridor represents the 'safe' motion of the robot moving towards its goal.

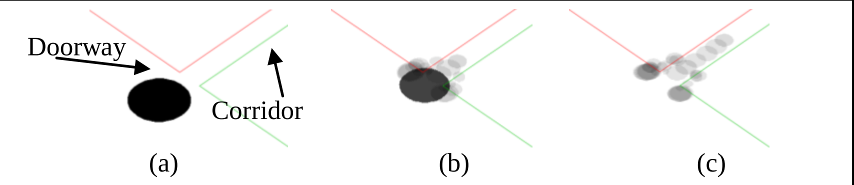


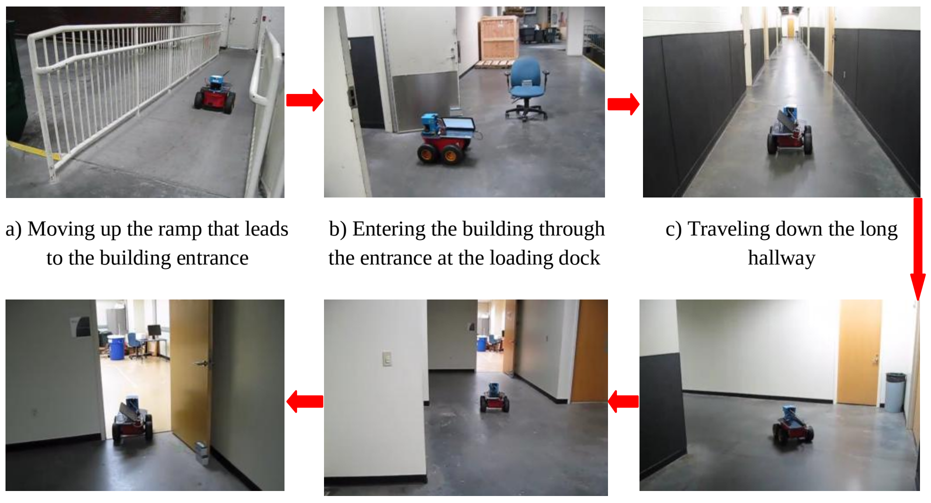
Fig. 11. Position distribution during traversal (a-c) of a door and corridor.

6 Results

We conducted a validation of our performance predictions for the multiple waypoint mission described in Section 3 using the following method. The VIPARS module was used to generate a prediction of the robot position after completing the mission. The robot motion and sensing uncertainty distributions used in VIPARS were calibrated for the Pioneer 3-AT robot for an indoor surface. This calibration is described in [16]. The robot mission was carried out a number of times and measurements made of the robot's success at completing the mission. The prediction and validation results were then compared. In [16] this same approach was used, and the accuracy of a set of single move missions was successfully validated. The validation procedure used in this paper is described in more detail in Section 6.1, the validation results reported in Section 6.2 and their analysis presented in Section 6.3.

6.1 Validation Procedure

The multi-waypoint mission was carried out with a Pioneer 3-AT robot as shown in Figure 12. The mission area is approximately *60×20* meters. The robot started at the bottom of the ramp. The start location of the robot is *(8.40, 23.80)* with respect to the world coordinates as shown in Figure 2. The waypoints for the mission are *(18.20,* *23.80)*, *(18.0, 20.80)*, *(58.75, 22.50)*, *(58.75, 33.75)*, and *(60.50, 40.50)*; and the robot is to visit the waypoints in the order listed with *(60.50, 40.50)* as the final waypoint. Following the waypoints, the robot moved up the ramp which leads to the loading dock where an entrance to the building is located. The robot then entered the building and traveled down a long hallway (approximately *40* meters in length), which leads to the room of interest located at the end of the hallway. The performance criterion for the mission is whether the robot had gained access to the room of interest (i.e., reached the final waypoint, which resides in the room). The mission was run *40* times and the numbers of successes and failures were recorded. The result is shown in Table 2. Most failures observed were due to the robot being stuck at the corner near the third waypoint as in Figure 12d. The reason for the failure is that the robot was not able to reach the third waypoint at the end of the long corridor. While the robot was near the waypoint physically, its internal localization said otherwise due to error accumulation in the odometry. As a result, the robot kept trying to go the third waypoint, but the corner walls prevented it from going anywhere.



a) Moving up the ramp that leads to the building entrance

b) Entering the building through the entrance at the loading dock

c) Traveling down the long hallway

f) Entering the room with potential biohazard threat

e) Moving toward the room entrance

d) Rounding a corner

Fig. 12. Snapshots of Pioneer 3-AT carrying out the mission presented in Fig. 3.

Table 2. Validation Result

# of Runs	# of Failures	# of Successes	P(Success)
40	12	28	70%

6.2 VIPARS Prediction

The Missionlab FSA is manually translated to set of PARS equations. Our ultimate objective is to automate this translation, but our first step is manual translations by which we are building experience required to specify the automatic translator. The waypoint mission of Section 3 is approximated in PARS as:

$$\begin{aligned}
&\textbf{Mission}_{g1,g2,g3,g4}(p,hi)(v,ho) = \\
&\qquad \textbf{Turn}_{g1}(p,hi)(ho)\ ;\ \textbf{MoveToVC}_{g1}(p)(v)\ \ ; \\
&\qquad \textbf{Turn}_{g2}(p,hi)(ho)\ ;\ \textbf{MoveToVC}_{g2}(p)(v)\ \ ; \\
&\qquad \textbf{Turn}_{g3}(p,hi)(ho)\ ;\ \textbf{MoveToVC}_{g3}(p)(v)\ \ ; \\
&\qquad \textbf{Turn}_{g4}(p,hi)(ho)\ ;\ \textbf{MoveToVC}_{g4}(p)(v)\ \ ; \\
&\qquad \textbf{Turn}_{g5}(p,hi)(ho)\ ;\ \textbf{MoveToVC}_{g5}(p)(v)\ \ .
\end{aligned}$$

The mission is five instances of a process that turns the robot to face the goal $\textbf{Turn}_{g1}$, and a process that then moves the robot towards that goal, $\textbf{MoveToVC}_{g1}$. Note that this network also include port connection information (as in e.g., eq (1)), which we omitted for brevity in previous sections. This information specifies the connections for the position input (p), the heading input (hi), the heading output (ho) and the velocity output (v). The system process is the concurrent, communicating composition of the mission and environment processes:

$$\textbf{SYS}= \textbf{NEnv}_{P0,H0}(c2,c3)(c1,c4) \mid \textbf{Mission}_{G1,G2,G3,G4}(c1,c4)(c2,c3)\ .$$

The capital letter parameters *P0, H0, G1,G2* and so forth are the initial conditions for the system: the initial position, heading, goal locations etc. The port connections *c1,...,c4* connect the position, heading and velocity ports on the mission to those in the environment model. The **NEnv** process is similar to **Env** in eq. (5), but with the information about heading and rotational uncertainty included. The process contains *no information* about walls or laser sensing to detect and respond to walls and obstacles. We have included this kind of information in previous work (e.g., Figure 11), so the approach will handle it effectively, but our objective is to not require an accurate map, or even any map, for verification, since that information may not be available. We are developing an approach to include instead information about the density of walls/obstacles in the environment which produces useful preformance results without needing an accurate map.

The VIPARS module first determines if **SYS** is composed of purely TR processes. If so, it can be verified by determining if a system period exists, and if one does, by extracting the system flow functions and using the DBN filtering approach presented in Section 5.3. If **SYS** is not composed of purely TR processes (as in this case), then the result presented in Section 5.4 is used to break up the system into a sequence of networks of purely TR processes, and the DBN filtering applied to each in turn. In this example, 10 such networks are extracted and filtered in sequence. The goal of reaching the final location is applied to each filtering result.

In [16], we show how this goal is specified and matched with the **SYS** network to determine what variables to inspect during filtering. In this case, the final robot location is inspected on each filtering step. When the cumulative probability of the robot being at the goal location has surpassed a constant threshold P_{min}, then verification terminates and reports the performance criterion met. If filtering continues to step T_{max} (a prespecified mission constant, the maximum allowed time) and the cumulative probability of the robot being at the goal location is still less than P_{min} then verification ends and reports the performance criterion not met. The results in [16] demonstrate statistically the predictive power of this approach for single move missions. However, most waypoint missions will have many moves, and that is the more complex case presented here.

VIPARS reported a successful verification for this mission with final position distributions (in *mm*) shown in Table 3. We ran VIPARS several times with different P_{min} to determine a maximum value for a successful verification (i.e., P_{max} = largest

P_{min} before T_{max}). These are shown as the last column in Table 3. Since a failure could occur at any waypoint, we estimate the probability for success as the product of success probabilities at each waypoint: P_{succ}= 0.91*0.99*0.81*0.99*0.99=71.5%. The lowest P_{max} was for the third waypoint, with $P_{max}=81\%$.

Table 3. VIPARS Waypoint Distributions

W#	**(μ_x, μ_y)**	**Σ**	**P_{max}**
1	*(17468, 23585)*	*[2610, 0; 0,8830]*	*0.91*
2	*(17850, 21206)*	*[4675, 286; 286, 9449]*	*0.99*
3	*(59411,21639)*	*[14986, -608; -608, 48005]*	*0.81*
4	*(59092,33444)*	*[24717, -218; -218, 50625]*	*0.99*
5	*(60422, 39764)*	*[30051,-1048; -1048, 52273]*	*0.99*

6.3 Comparison of Predicted and Measured Results

Empirical experiments show a success probability of 70% for this mission, given 40 runs with 12 failures. Our predicted success rate is ~72%. We can statistically compare both predictions with the validation results using a *z-statistic* proportion test. The null hypothesis is H_0: $p_{succ}=0.72$ and H_a: $P_{succ}<0.72$. For applicability of the test, we need to ensure $np_0=40\times0.72>10$. We calculate the z-statistic as $z=-0.28$, and $p(Z<-0.28)=0.3897$ from the standard distribution tables.

$$z = \frac{p_1 - p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}} = \frac{0.7 - 0.72}{\sqrt{\frac{0.72(1-0.72)}{40}}} = -0.28$$

Since $0.05<<0.3897$ we (emphatically) fail to reject H_0: $p=0.72$ at the 95% confidence level. So although our predicted results are a little more optimistic than the experimental results, they are not significantly different. The waypoint with lowest P_{max} is also the one that offered most difficulty during empirical validation, and this also supports the usefulness of the VIPARS prediction.

However, they do differ and it begs the question why? Since the PARS environmental model does not include any walls or sensing, we conclude that its more optimistic result is reasonable. The result of including walls in the model, e.g., as in [16], would be to break the unimodal probability result (23) into a multimodal, Mixture of Gaussians, and for some of the components to relate to actual collisions along the route, reducing the overall probability of successful completion (e.g. as in Figure 11). In fact, our planned inclusion of a wall-factor will leverage exactly this phenomenon.

7 Conclusions

We have presented a novel approach to verification of performance guarantees for behavior-based robot programs. The approach differs from prior work in its avoidance of the concept of state via the use of a PA framework. The general case of software verification runs afoul of the halting problem. To address this fundamental limitation, most work therefore focuses on specific cases; we have focused on a PA structure that we believe captures behavior-based programming well: concurrent interacting systems of TR processes. TR processes have the useful feature that they easily allow the construction of recurrent flow-functions that capture how the TR processes transform variable values on each recursive step. The *SysGen* algorithm constructs a single *system period process* from the periods (the recursive bodies) of each component process, if one exists. The SysGen result was extended here to include an additional process in the system that may be a sequential mixture of multiple straight line and TR processes.

We present an algorithm, *FloGen*, that extracts the flow-function for the system period by following and resolving communications over port connections between the processes in the system period. To model uncertainty, which is a sine qua non for realistic robot results, we extend the PA to allow processes to have random variables. We show that the flow function in this case can be mapped to a Bayesian Network, and the recurrent nature of the flow-functions can be captured as a Dynamic Bayesian Network. The verification problem for the random variable case can then be phrased as a DBN filtering problem.

Prior work [16] reported a validation of a single move for a Pioneer 3-AT robot in indoor conditions at various velocities. The results show strong statistical evidence of the predictive power of the approach. In this paper, we extend that validation to a multiple waypoint mission. We do not validate the accuracy of the resulting location in this case, but rather the accuracy of the prediction of success. Empirical testing of this mission yielded a 70% success probability. The VIPARS prediction, employing the Pioneer 3-AT calibration from [16], was 72%. The environmental model used in VIPARS did not include walls or wall sensing, which meant the results were more optimistic than practice. Discounting this, the results are certainly sufficiently in agreement to count the valiation as successful.

A principal cause of the discrepancy is the effect of walls and doorways on the motion of the robot. In prior work we showed how a Mixture of Gaussian random variable representation allowed us to represent well the interaction between the robot and walls, doorways, corners, etc. So our method can include information that is known about the environment. However, we do not want our approach to be tied to the necessity for accurate, metric map information, since that may not be available for a C-WMD mission. Instead, we are approaching this uncertainty with a *wall-factor* value that represents how walled-in the environment is: an open warehouse (low wall-factor) versus office corridors (high wall-factor).

Although a C-WMD mission might have some waypoint aspects, it is more likely that the mission will be of the explore-and-find nature rather than strictly follow-the-waypoints, and will involve multiple robots. We are already specifying and executing missions of this kind in *MissionLab* and we will now study how VIPARS can be used to verify performance guarantees for these missions. We anticipate that the extended SysGen approach will be used to switch between behavioral-states for these kind of missions, rather than the simpler way it was used here, to switch from one waypoint to the next. It will also require characterization of target identification sensors in a manner similar to how the Pioneer 3-AT motion uncertainty was calibrated.

References

1. Arkin, R.C., *Behavior-Based Robotics*, MIT Press, Cambridge, MA, 1998.
2. Arkin, R. C., Lyons, D., 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. *Proceedings of SPIE*. Vol. 8387. 2012.
3. Baeten, J., A Brief History of Process Algebra. *Elsevier Journal of Theoretical Computer Science – Process Algebra*, 335(2-3), 2005.
4. Boem, C. and Jacopini, G., Flow diagrams, Turing machines and languages with only two formation rules, CACM 9(5) 1966.

5. Casper, J., & Murphy, R. R., Human-robot interactions during the robot-assisted urban search and rescue response at the world trade center. *Systems, Man, and Cybernetics, Part B: Cybernetics*, *IEEE Transactions on*, *33*(3), 367-385, 2003.
6. Clark, E., Grumberg, O., Peled, D., *Model Checking*. MIT Press 1999.
7. Hinchey M.G., and J.P. Bowen, *High-Integrity System Specification and Design*, FACIT series, Springer-Verlag, London, 1999.
8. Jhala, R., Majumdar, R., Software Model Checking. *ACM Computing Surveys,* V41 N4, Oct 2009.
9. Johnson, B., and Kress-Gazit, H., Probabilistic Analysis of Correctness of High-Level Robot Behavior with Sensor Error, *Robotics Science and Systems*, 2011.
10. Karaman, S., Rasmussen, S., Kingston, D., Frazzoli, E., Specification and Planning of UAV Missions: A Process Algebra Approach. *2009 American Control Conference*, St Louis MO, June 2009.
11. Kress-Gazit, H., Pappas, G.J., Automatic Synthesis of Robot Controllers for Tasks with Locative Prepositions. *IEEE Int. Conf. on Robotics and Automation*, Anchorage, Alaska, May 2010.
12. Labinaz, G., Bayonmi, M., and Rudie, K., Modeling and Control of Hybrid Systems: A survey, *IFAC 13th World Congress*, 1996.
13. Lyons, D., Arkin, R., Towards Performance Guarantees for Emergent Behavior. *IEEE Int. Conf. on Robotics and Automation*, 2004.
14. 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.
15. Lyons, D., Arkin, R., Nirmal, P and Jiang, S., Designing Autonomous Robot Missions with Performance Guarantees. *Proc*. *IEEE/RSJ IROS, Vilamoura Portugal,* Oct. 2012.
16. Lyons, D., Arkin, R., Nirmal, P and Jiang, S., Liu, T-L., A Software Tool for the Design of Critical Robot Missions with Performance Guarantees. *Conference on Systems Engineering Research (CSER'13)* Atlanta GA, 2013.
17. MacKenzie, D., Arkin, R., Evaluating the Usability of Robot Programming Toolsets. *Int. Journal of Robotics Research*, Vol. 4, No. 7, April 1998, pp. 381-401.
18. MacKenzie, D., Arkin, R.C., Cameron, R., Multiagent Mission Specification and Execution. *Autonomous Robots*, Vol. 4, No. 1, Jan. 1997, pp. 29-52.
19. MacKenzie, D.C., *Configuration Network Language (CNL) User Manual*. College of Computing, Georgia Tech, V. 1.5, June 1996.
20. Napp, N., Klavins, E., A Compositional Framework for Programming Stochastically Interacting Robots, *Int. Journal of Robotics Research* 30:713 2011.
21. Ramadge R.J., and Wonham, W.M., 1987. *Supervisory control of a class of discrete event processes*. SIAM J. Control and Optimization, 25(1), pp. 206-230.
22. Shenoy, P.P., Inference in Hybrid Bayesian Network Using Mixtures of Gaussians, *22nd Int. Conf. on Uncertainty in AI*, Cambridge MA 2006.
23. Steenstrup, M., Arbib, M.A., Manes, E.G., *Port Automata and the Algebra of Concurrent Processes*. JCSS 27(1): 29-50 (1983).