Getting it Right the First time: Robot Mission Guarantees in the Presence of Uncertainty*

D.M. Lyons, R.C. Arkin, P. Nirmal, S. Jiang, T-M Liu, J. Deeb

Abstract—**Certain robot missions need to perform predictably in a physical environment that may only be poorly characterized in advance. We have previously developed an approach to establishing performance guarantees for behavior-based controllers in a process-algebra framework. We extend that work here to include random variables, and we show how our prior results can be used to generate a Dynamic Bayesian Network for the coupled system of program and environment model. Verification is reduced to a filtering problem for this network. Finally, we present validation results that demonstrate the effectiveness of the verification of a multiple waypoint robot mission using this approach.**

I. 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 only 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 regarding whether or not the mission in question 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 prior work [2][13]-[17] using the Georgia Tech *MissionLab* toolkit [18]-[20] we translate mission software to a process algebra framework, and we verify whether the mission software when executed in an operator-selected physical environment model (also process algebra) will achieve an operator-specified performance guarantee. The *SysGen* algorithm [15] identifies periodic behavior in a set of concurrent, connected processes that represent a behavior-based robot program and its environment. The output of *SysGen* is a set of recurrent parameter flow functions. 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 generated. 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. Finally, new results are presented for a multi-waypoint robot mission and we validate these results showing the predictive power of our method.

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

D.M. Lyons, P. Nirmal and T-M Liu are with the Dept. of Computer & Information Science, Fordham University, Bronx NY 10458, USA (Ph: 718-817-4485, Fx: 718-817-4488, Em: dlyons@cis.fordham.edu).

R.C. Arkin, S. Jiang and J. Deeb are with the Mobile Robotics Laboratory, Georgia Institute of Technology, GA 30332, USA (Em: arkin@cc.gatech.edu).

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

II. 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. A completely general solution is ruled out by the undecidability of the halting problem; however, much research has been conducted on restricted instances of this problem. *Model checking* [6]-[8] is a very successful technique in which a program is mapped to a *Kripke* system – a state-based transition system where states are labeled with sets of propositions,. 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 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.

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 model in an ongoing fashion. increasing the state-space by the product with the environment model. A second characteristic is that there may be a necessary continuous nature to some aspects of the environment; not easily handled with model-checking since the state space will grow with the size of the value space of the variables. A program with ten floating point 32-bit numbers has a potential state-space of size > 10^{90} for example. Finally, significant uncertainty pertains to the result of robot sensing and motion; this cannot be ignored or the results are not realistic.

A state-based approach experiences significant combinatorial problems due to the characteristics discussed in the previous paragraph. However, model-checking is not the only approach to software verification. *Satisfiability Modulo Theories* (SMT) [21] cast verification as the satisfiability of expressions in a set of theories that can include real-numbers, array references and most recently recursive functions. The challenge is in automatically transforming a program into an appropriate collection of expressions.

In [15] we introduce a process algebra (PA) approach to representing robot programs and environment models. The advantage of PA is that it can be used to determine how a process transforms its inputs to produce outputs without reference to states. 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 earlier work in PA for performance analysis of robot programs [13] leveraged the trace, or history of events, of a process. In this paper, we use a PA that includes I/O port communications [24] and leverage this structural locality information.

We focus on 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 percept is responded to, the robot may remain in this behavioral state or move to another that handles a different set of percepts. For the specific case of a system of a PA environment model and programmed behaviors represented in *tail-recursive* process definitions, we proposed a novel process interleaving theorem *SysGen* that allowed us to identify a single composite *system period process*. This process contained all the port-to-port communications that could happen in the system as part of the percept-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 parameter flow-functions, since they related the value of variables in one iteration of the system period to that in the next iteration. The verification problem in this framework is the satisfiability of these functions modulo recurrent functions and real-numbers.

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 [22] 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], addressing the automatic controller generation problem rather than verification, develop a model-checking based algorithm that handles uncertainty based on Discrete Time Markov Chains; however, they comment on the intractability of their approach for large state spaces.

In this paper, we extend the flow-function approach to include random variables, and we map the solution of a system of flow-functions to a filtering problem for a Dynamic Bayesian Network. This approach can also leverage various parametric uncertainty distributions, including Mixture of Gaussians [23], to capture motion and sensor uncertainty.

III. 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 Afghanistan). 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 desirable 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 significant challenge that is different from traditional software verification – the robot has to work in the real world, and the real world is inherently unpredictable.

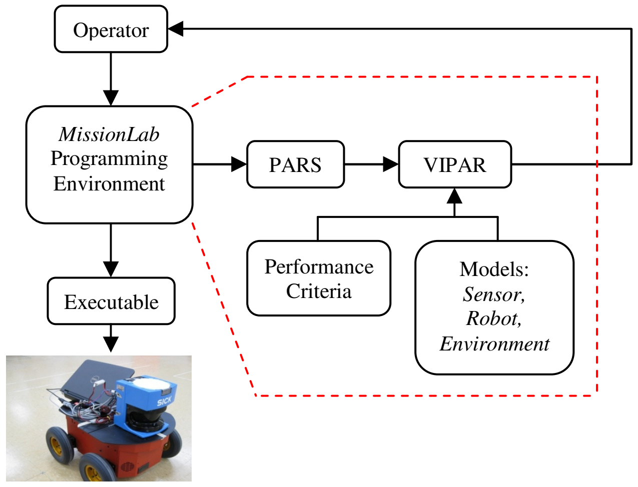

Operator
MissionLab Programming Environment
PARS
VIPAR
Performance Criteria
Models: *Sensor, Robot, Environment*
Executable

Fig. 1. *MissionLab* robot mission specification toolset with VIPARS verification module, see [2] for more details.

We have built our robot mission verification framework upon *MissionLab*[2], a behavior-based robot programming environment [19]. *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* by which missions can be verified before the executable generation step [2].

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, VIPARS (Verification in Process Algebra for Robot Schemas). To initiate the verification of a mission, the robot program is compiled from CNL (*MissionLab's* internal representation [20]) to PARS (Process Algebra for Robot Schemas), the language understood by VIPARS. The robot operator also needs to provide VIPARS with models of the robot, the sensors it is equipped 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

[2]*MissionLab*, now in version 7.0, is freely available for research and education at http://www.cc.gatech.edu/ai/robot-lab/research/MissionLab/.

guarantee for the mission based on how well the specified 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.

A. *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.

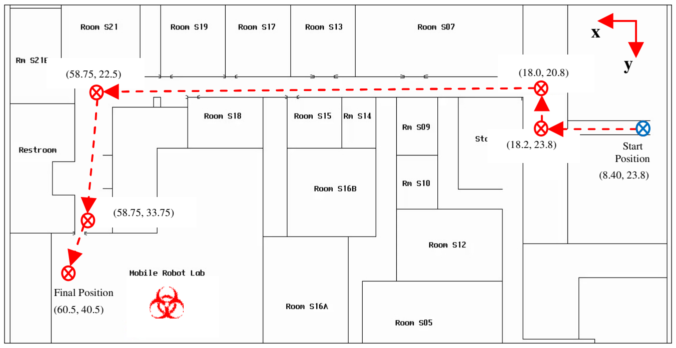


Fig. 2. Building layout with waypoints labeled in red

The program for the multi-waypoints mission from Fig. 2 is shown in Fig. 3; this behavioral FSA 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 velocity *dropoff_radius* around the goal location. *Dropoff_radius* specifies the distance from the goal, where the robot starts to slow down as it approaches the goal location. The *AtGoal* trigger causes a transition 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 a simulation is insufficient to provide performance guarantees for the mission.

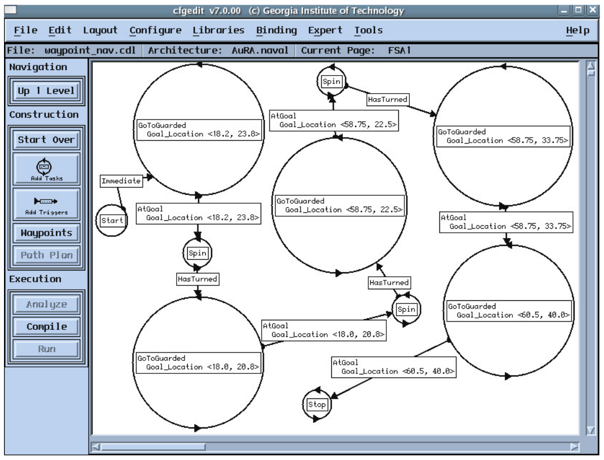
B. *Mission Performance Criteria*

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 probability that the system will perform under these criteria is the mission effectiveness. Quantification of this probability provides a metric of success for the system, which in turn allows for decision-makers to properly assess the different options associated with each mission. Mission effectiveness (*ME*) is calculated as follows: $ME = A \times R$, where A is the availability at the start of the mission and R is the mission reliability with environment and operator effects included [12]. For the purposes of this paper, we will define availability as the probability that an item will be operational at a given time, and reliability as the ability of a system to operate under designated performance criteria for fixed periods of time. This model is consistent with the model previously provided in [2] where the first term is effectively the availability and the second reliability.

Previous models for mission effectiveness rely entirely on empirical test data. This makes characterizing a system's effectiveness difficult when no such previous data exists, which is often the case with robotic systems. Gathering data specific to each C-WMD mission is not really feasible, so predicting the rates of failures for the system becomes imperative. Since both availability and reliability are characterized by the behaviors of the failure rates, if the failure rates of the components in the system can be predicted without the use of past data then the mission effectiveness can also be predicted.

C. *Verification of Performance Guarantee*

Designs rarely work the first time off the drawing board. 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 complete the mission, we need to have some guarantee that our robotic system will succeed before its deployment.



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,* showing the mission design for a multiwaypoint mission. Each 'state' and 'trigger' node (the large and small circles) corresponds to a behavior and the arrows denote sequencing.

To obtain a performance guarantee for the robot mission in Fig. 3, the operator needs to compile the mission into PARS, select models of sensor, robot, and the environment, and provide VIPARS with the performance criteria (Fig. 1). Currently, VIPARS outputs 1) a Boolean answer regarding whether the mission as specified can be successful, and 2) a list of 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.

IV. PARS Representation of Missions

PARS is a process algebra [4] 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. For a more thorough introduction, and a wider selection of controllers and environment models, see [13][17].

The semantics of a process in PARS is an extended port-automaton [24], an automaton equipped with communication ports over which it can send messages to other concurrent automata, and extended with duration and partitioned end-states (success/stop and fail/abort). A process **P** with initial parameter (variable) values *u1,u2,…,un,* input ports/connections *i1,i2,…,im,* output ports/connections *o1,o2,…,op* and final result values *v1,v2,…,vq* is written as:

$$\mathbf{P}_{u1,u2,\ldots,un}\,(i1,i2,\ldots,im)\;(o1,o2,\ldots,op)\;\langle v1,v2,\ldots,vq\rangle \tag{1}$$

Subscripts indicate parameters. Parentheses are used to group port connections and the angle brackets, variable results. In this paper (except for Section VI, which refers to actual implementation) we will just use global port names rather than the more general but more verbose port-to-port connections. For brevity, the parts of a process description that are empty are typically omitted. Process variables (initial parameters, results) can be random variables; we'll return 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 (see 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:

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

A sequential composition (;) in which the first process

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 y #
$\mathbf{Eq}_{a,b}$, $\mathbf{Neq}_{a,b}$, $\mathbf{Gtr}_{a,b}$, etc.,	$a{=}b$, $a{\neq}b$, $a{>}b$, etc.,	Otherwise

ends in abort (see Table 1) just aborts. This implements a conditional construct. Other composition operations include parallel-max (|), a parallel communicating composition of processes that terminates when all have terminated, and parallel-min or disabling (#), a parallel communicating composition of processes that terminates as soon as any terminate. 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 repetitive construct. Any language that implements sequence, condition and loop constructs is sufficient to represent any program [5]; 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*.

A. PARS Controllers

One objective of our project is to automatically translate *MissionLab*'s underlying CNL mission specification language [20] 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 III, 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 $\mathbf{Mission}_{\boldsymbol{w},0}$ 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*.

The example used in this paper is basically a motion example. The representation and method is not however intrinsically limited to this controller or to motion examples.

B. 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:

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

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* (where *r* is a random variable). The process **Odo** (short for Odometry sensor) makes position information (with noise) available in a loop until terminated by the **Delay** enforcing the discrete progress of time in steps of at most *t.* 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 $\Phi \sim N(\mu_m, \sigma_m)$.

C. 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. Another approach, called *refinement*, is to use the same language for property and program, but consider the property to be a more abstract version of the program. We specify performance goals directly in PARS. For example, the designer may wish to specify that the robot arrives at position *a* after time *t1* and stays there for at most a time *t2*:

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

where *t1* and *t2* are variables. A property specification process network 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.

V. Verification Method

The verification approach presented at the end of the last section is as follows: Given a parallel composition of a controller and system:

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

will (7) achieve the performance specification in (6)? In [15] we leverage a property of behavior-based systems to reduce the complexity of this problem. We present an algorithm, *SysGen* that matches the recurrent structure in the controller and environment processes 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, flow-functions, one function 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. We consider in this paper a practical Bayesian approach to the solution of these flow functions.

```
FloGen( FS = {f1,…fm} ): // component flow fns 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←port in pi that generated $v_j$
4.         While (a !=⊥)
5.             cm(a) is the network connection of a on pk
6.             u←parameter value to the port operation on cm(a)
7.             a←port in pk that generated u or ⊥ if none
8.         Replace $v_j$ with u
```

Fig. 4. Flow Function Generation Algorithm, *FloGen*

A. 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 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 for two processes.

The *FloGen* algorithm (Fig. 4) produces a flow function that includes these transformations for each parameter to the system period. For each flow variable, $r_i \in R = \{r_1,\ldots,r_n\}$, *FloGen* traces its transformation through processes and port communications to generate a single flow function f_i defined as:

$$f_i(r_1,\ldots,r_n):\ \mathrm{val}(r_{1,k}) \times \ldots \times \mathrm{val}(r_{n,k}) \rightarrow \mathrm{val}(r_{i,k+1}) \tag{8}$$

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*.

Flow variables may contain 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

$$P(\ r_{i,k+1} \mid R_k)\ = f_i(\ R_k). \tag{9}$$

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:

$$P(\ r_{i,k+1} \mid R_{i,k})\ = f_i(\ R_{i,k}),\ R_{i,k} \subseteq R_k \tag{10}$$

The resulting structure can be drawn as a Bayesian network as shown in Figure 6.

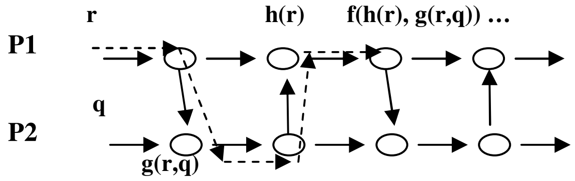


Fig. 5. Example of variable value transformation (dotted lines) for variables *r* and *q* in a single system period composed of two processes P1 and P2.

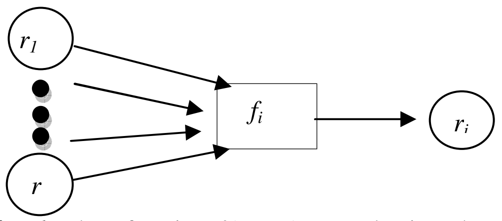


Fig. 6. Flow function $f_i(r_1 \ldots r_n) = r_i$ evaluation shown as a Bayesian Network.

As long as flow functions can include the effect of program conditionals [17], 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). We define a function *F* as the transition model of the DBN, where

$$F(R_k) = (\,f_1(R_{1,k}), f_2(R_{2,k}),\ldots) \tag{11}$$

B. Verification as Filtering

The process of matching the system and goal networks [15] identifies a subset of the flow-variables, $G \subseteq R$, and the values to be associated with them

$$GV=\{\ (g,v)\mid g\in G \text{ and } v\in val(g)\ \} \quad (12)$$

The verification problem asks whether the execution of the controller in the given environment will result in 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 of the value v. Our definition of a successful verification is that

$$P(GV_k \mid R_{1:k}) > P_v \quad (13)$$

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 to step k. We introduce an observation model $GF(R_k)$ to implement this evaluation at any step:

$$GF(R_k) \;=\; P(GV_k \mid R_k) \quad (14)$$

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

$$P(GV_k \mid R_{1:k}) = \sum_{i=1}^{k} P(GV_i \mid R_i)P(R_i \mid R_{1:i-1}) \quad (15)$$

Since each R_i is linked to the previous by the transition model $R_{i+1} = F(R_i)$, and goal satisfaction is related to R_i by the observation model $GF(R_i)$:

$$\begin{aligned} P(GV_k \mid R_{1:k}) &= \sum_{i=1}^{k} P(GV_i \mid R_i)\prod_{j=1}^{i} P(R_j \mid R_{j-1}) \quad (16)\\ &= \sum_{i=1}^{k} P(GV_i \mid R_i)F^{i}(R_1)\\ &= \sum_{i=1}^{k} GF(F^{i}(R_1)) \end{aligned}$$

While thresholding on P_v in (13) 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:

$$P(GV_k \mid R_{1:k}) > P_v \text{ and } k<K_{max} \quad (17)$$

This solution is reasonable 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).

C. *Extension of SysGen*

Consider a a system **Sys** composed of set of processes **P1**, **P2**, ..., **Pm**:

$$\begin{aligned} \mathbf{Sys} &= \mathbf{P1} \mid \mathbf{P2} \mid \ldots \mid \mathbf{Pm} \quad (18)\\ &= S(\mathbf{P1}, \ldots, \mathbf{Pm})\ ;\ \mathbf{Sys} \end{aligned}$$

where each $\mathbf{Pi} = \widehat{\mathbf{Pi}}\ ;\ \mathbf{Pi}$, and where $\widehat{\mathbf{Pi}}$ is not recursive; that is, each **Pi** is a tail-recursive (TR) process. *SysGen* constructs the system period S(**P1, ..., Pm**) for such systems. However, *Sysgen* is defined in [15] *only* for a composition of TR processes. If one of the processes, **Pi**, is for example a sequence of $\mathbf{Goto}_a$ processes in a waypoint mission, then **Pi** is not TR. There is a straightforward extension for *SysGen* to automate this. Let one process **Pi** in (18) be non-TR, then let us consider all the scenarios that can result, and let $\widehat{\mathbf{Pi}}$ be the 'period' we then identify for use in *SysGen*:

1. **Pi** is pure straight-line code: In that case, we have $\widehat{\mathbf{Pi}} = \mathbf{Pi}$; so we calculate the system 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. **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 with **Pi** replaced by **SL**, calculating the flow-function and DBN and filtering for one time step, and *carry the variable values over to a second system* where
   (b) we address the system with **Pi** replaced by **T**, which is TR and can be handled in the normal fashion.
3. **Pi** is a sequence of two TR processes, $\mathbf{Pi}_{a,b} = \mathbf{T1}_{a}\langle y\rangle\ ;\ \mathbf{T2}_{a,b,y}$ or a TR process followed by straight-line code. We also break this into a sequence of two problems with **Pi** replaced by **T1** in the first and **Pi** replaced by **T2** in the second, carrying the variable values over between both problems.

Using this approach, a mission with k sequential motions will be broken automatically into k filtering problems. In the current version of VIPARS, each problem is treated as a distinct and independent step, and the probability of success is simply the product of the step probabilities.

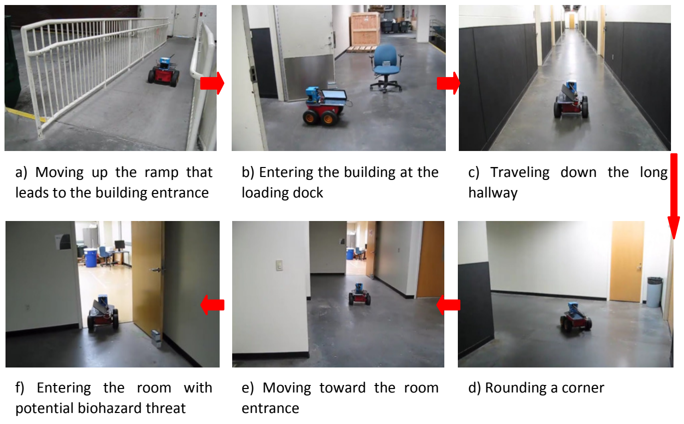

a) Moving up the ramp that leads to the building entrance
b) Entering the building at the loading dock
c) Traveling down the long hallway
f) Entering the room with potential biohazard threat
e) Moving toward the room entrance
d) Rounding a corner

Fig. 7. Snapshots of Pioneer 3-AT robot at several points during a validation of the multiple waypoint mission presented in Fig. 3.

Table 2. Validation Result

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

VI. Results

We conducted a validation of our performance prediction for the multiple waypoint mission (Section III). 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 [16]. The robot mission was carried out 40 times and measurements made of the robot's success at completing the mission. The prediction and validation results were then compared. In [16] this approach was used, to validate the accuracy of a set of single waypoint missions.

A. Validation Procedure

The multi-waypoint mission employed a Pioneer 3-AT robot (Fig. 7). 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 missions 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 the loading dock ramp 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 objective of this guided-navigation mission is to have the robot enter a room where potential biohazards may be stored. Once the robot is in the room, it could deploy its onboard sensors (e.g., chemical) to search the room for biohazards. However, this paper focused on verifying the guided-navigation mission, thus the mission is considered successful once the robot enters the room of interest. Verification of the biohazard detection and identification mission with chemical sensors will be conducted in our future work.

The mission was run *40* times and the number of success/failures was recorded (Table 2). Most failures observed were due to the robot being stuck at the corner near the third waypoint as in Figure 7d. 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 motion. Obviously better results could be obtained with better sensors, but that is not the point of this paper: it is rather to predict the likelihood of success given a particular robot configuration, hence the reason we encountered such a high failure reason.

B. VIPARS Prediction

The *MissionLab* FSA is manually translated to set of PARS equations. The waypoint mission of Section III is approximated in PARS as:

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

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

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

The parameters *P0, H0, G1, G2* and so forth in the expression above are the initial conditions for the system: the initial position, heading, goal locations etc. The port connections *c1,...,c4* in the expression 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., [16],[17]), but our objective eventually is to not require an accurate map, or even any map, for verification, since that information may not be available.

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 (eq. (18)), and if one does, by extracting the system flow functions and using the DBN filtering approach presented in Section V.*B*. If **SYS** is not composed of purely TR processes (as in this case), then the result presented in Section V.*C* is used to break up the system into a sequence of networks of TR processes, and the DBN filtering applied to each in turn. In this example, 10 networks are extracted and filtered in sequence. The goal of reaching its final location is applied to each filtering result.

In [15], we show how this goal is specified and matched with the **SYS** network to determine what variables to inspect during filtering. The termination condition for filter is shown in eq. (17). The results demonstrate statistically the predictive power of this approach for single waypoint missions. Most waypoint missions have many waypoints, and that is the more complex result presented here.

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*

VIPARS reported a successful verification with final position distributions (in *mm*) shown in Table 3. Calculation time on an Intel Core 2 Duo 1.8GHz laptop was of the order of a few minutes including (overly) extensive diagnostic output. VIPARS was run several times with different P_{min} to determine a maximum value for a successful verification (i.e., 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 estimate for success as the product of successes at each waypoint, p_{succ} = 0.91*0.99*0.81*0.99*0.99=71.5%. The VIPARS breakdown into these 5 subproblems is automatic and just separated out here for a more detailed comparison with the validation.

C. Comparison of Predicted and Measured Results

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

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.

VII. Conclusions

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 captures behavior-based programming well and avoids explicit state: 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. To model uncertainty, which is a sine qua non for realistic robot results, processes are allowed 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 a DBN filtering problem.

Lahijanian et al. [11] and Johnson & Kress-Gazit [9] address the problem of automatically generating a controller from a high-level specification in a temporal logic such as LTL or CTL. Model-checking provides techniques to verify an automaton with respect to a temporal logic specification and hence is leveraged in the aforementioned and similar formal methods work in robotics. Our focus is verification of operator-generated mission software to provide performance guarantees and hence the temporal logic aspect is not as useful an ingredient. Because of state-combinatorics, we have elected to follow an SMT-like approach instead.

Sampling approaches, such as *Ymer* [25], use Monte Carlo sampling of execution paths to verify probabilistic systems. Simulation is typically used to generate sample paths and sufficient samples are taken to verify a system within a bound for false positives and a bound for false negatives. Our approach is parametric rather than sampling based and does not need bounds, and of course, does not need to carry out multiple simulation runs. An advantage of sampling methods is that they can handle Semi-Markov or Generalized Semi-Markov systems.

In [16] we reported strong statistical evidence of the predictive power of our approach for single motions at various distances and speeds. Here, we extended that validation to a multiple waypoint mission. Empirical testing of this mission on a Pioneer 3-AT robot yielded a 70% success probability. The VIPARS prediction was 72%. The results are statistically strong enough to count the validation as successful.

Although a C-WMD mission might have some waypoint aspects if sufficient knowledge is available a priori, it is more likely that the mission will be of an 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.

References

[1] Arkin, R.C. (1998) *Behavior-Based Robotics*, MIT Press.

[2] Arkin, R. C., Lyons, D., Jiang, S., Nirmal, P., &Zafar, M. (2012) 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.

[3]Baier, C., et al. (2010) Performance Evaluation and Model Checking Join Forces *CACM* (53)9: 78-85.

[4] Baeten, J. (2005) A Brief History of Process Algebra. *Elsevier Jour. Theoretical Comp. Sc. – Process Algebra* 335(2-3).

[5] Boem, C. and Jacopini, G., (1966) Flow diagrams, Turing machines and languages with only two formation rules. *CACM* 9(5): 366-371.

[6] Clark, E., Grumberg, O., Peled, D. (1999) *Model Checking*. MIT Press.

[7] Hinchey M.G., & J.P. Bowen (1999) *High-Integrity System Specification and Design* FACIT series, Springer-Verlag, London.

[8] Jhala, R., Majumdar, R. (2009) Software Model Checking. *ACM Computing Surveys* 41(4) 21:53.

[9] Johnson, B., and Kress-Gazit, H. (2011) Probabilistic Analysis of Correctness of High-Level Robot Behavior with Sensor Error, *Robotics Science and Systems*, Los Angeles CA.

[10] Karaman, S., Rasmussen, S., Kingston, D., Frazzoli, E. (2009) Specification and Planning of UAV Missions: A Process Algebra Approach. *Amer. Control Conference*, St Louis MO..

[11] Lahijanian, M., Wasniewski, S., Andersson, S., Belta, C. (2012) Motion Planning and Control from Temporal Logic Specifications with Probabilistic Satisfaction Guarantees. *IEEE Trans. Rob.* 28(2): 396-409.

[12] Lie, Chang H*oon*, et al. (1984) Mission effectiveness model for a system with several mission types" *IEEE Trans. Rel.* 33.4 346-352.

[13] Lyons, D., Arkin, R. (2004) Towards Performance Guarantees for Emergent Behavior. *IEEE Int. Conf. on Rob. and Aut.* New Orleans LA.

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

[15] Lyons, D., Arkin, R., Nirmal, P and Jiang, S., (2013) Designing Autonomous Robot Missions with Performance Guarantees.. *IEEE/RSJ IROS,* Vilamoura Portugal.

[16] Lyons, D., Arkin, R., Nirmal, P and Jiang, S., Liu, T-L. (2013) A Software Tool for the Design of Critical Robot Missions with Performance Guarantees. *Conf. Sys. Eng. Res. (CSER'13)* Atlanta GA.

[17] Lyons, D., Arkin, R., Liu, T-L., Jiang, S., Nirmal, P. (2013) Verifying Performance for Autonomous Robot Missions with Uncertainty. *IFAC Intelligent Vehicle Symposium,* Gold Coast, Australia.

[18] MacKenzie, D., Arkin, R. (1998) Evaluating the Usability of Robot Programming Toolsets. *IJRR.* (4)7: 381-401.

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

[20] MacKenzie, D.C. (1996) *Configuration Network Language (CNL) User Manual.* College of Computing, Georgia Tech, V. 1.5.

[21] De Moura, L., Bjorner, N. (2011) Satisfiability Modulo Theories: Introduction and Applications. *CACM* 54(9): 54-67.

[22] Napp, N., Klavins, E. (2011) A Compositional Framework for Programming Stochastically Interacting Robots, *IJRR.* 30:713.

[23] Shenoy, P.P., (2006) Inference in Hybrid Bayesian Network Using Mixtures of Gaussians, *22nd Int. Conf. Uncertain. in AI*, Cambridge MA.

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

[25] Younes H., Simmons, R. (2002) Probabilistic verification of discrete event systems using acceptance sampling. *14th Int. Conf. on Computer Aided Verification* Copenhagen, Denmark.