Performance Verification for Robot Missions in Uncertain Environments[1]

D. M. Lyons*, R. C. Arkin‡, S. Jiang‡, M. O'Brien‡, F. Tang*, P. Tang*

*Fordham University
New York USA
dlyons@fordham.edu

‡ College of Computing,
Georgia Institute of Technology
Atlanta GA USA

***Abstract*— Establishing a-priori mission performance guarantees is crucial if autonomous robots are to be used with confidence in missions where failure could incur high costs in life and property damage. Automatic mission software verification, in addition to simulation and experimental benchmarking, is a key component of the solution for establishing performance guarantees. This component requires automatically verifying that the software constructed by the mission designer when executed in a partially known environment will adhere to the performance guarantee. In prior work we developed VIPARS, a unique approach to verifying performance guarantees for autonomous behavior-based robot software based on a combination of static analysis and Bayesian networks. While that approach produced fast and accurate verification of single robot missions with robot motion uncertainty, it did not address multiple-robot missions or any form of uncertainty related to environment geometry.**

This paper addresses the challenges involved in building a software tool for verifying the behavior of a multi-robot waypoint mission that includes uncertainly located obstacles and uncertain environment geometry as well as uncertainty in robot motion. An approach is presented to the problem of a-priori specification of uncertain environments for robot program verification. Two approaches to modeling probabilistic localization for verification are presented: a high-level approach and an approach that allows run-time localization code to be embedded in verification. Verification and experimental validation results are presented for several autonomous robot missions, demonstrating the accuracy of verification and the mission-specific benefit of localization.

Keywords-component; Probabilistic Verification, Validation, Multi-robot Missions, Behavior-Based Robots.

1 Introduction

It is crucial to be able to establish an a-priori guarantee of mission success for robots deployed in critical missions such as counter weapons of mass destruction (C-WMD) and other missions where failure brings serious consequences to life and property. In other, less critical applications it is highly desirable to have a-priori guarantees of performance to reduce overall mission costs. In prior work for the Defense Threat Reduction Agency (DTRA) [1], we have developed an approach to automatic verification of performance guarantees for autonomous behavior-based robot mission software operating in uncertain environments. We developed a unique combination of static analysis and Bayesian networks for efficient and automatic verification of performance guarantees for missions developed in the *MissionLab* [2] robot mission design toolkit, and demonstrated by experimental validation that the approach produced trustworthy results. While that work detailed the foundation of the approach, it only addressed the single-robot scenario, and it assumed operation in an open space, with no unexpected obstacles. This paper leverages that prior work [1] to also address the challenges of automatic verification of performance guarantees for single and multi-robot missions in environments with uncertain geometry.

Verification of robot software is related to general purpose software verification in its objective of taking a program as input and automatically determining whether that program achieves a desired objective or not [3]. It differs in that a robot program continually interacts with its uncertain and dynamic environment – which therefore must be included as part of the verification problem. In fact, this is rarely done in robot program verification and was one of the novel contributions of our prior work [1]; so, rather than addressing computational verification problems such as absence of deadlock or absence of run-time errors [4] [5] (important, but typically addressed in software verification), we have focused on establishing performance guarantees for the mission software with a complex and uncertain environment model. Also, like [6], we have focused on verification of behavior-based autonomous robots, a modular approach capable of robust performance in uncertain environments.

One contribution of this paper is an approach to the problem of a-priori specification of uncertain environments for robot program verification, in particular, to specifying an environment which may or may not contain obstacles with locations specified probabilistically. A consequence of this environment model is that verification must consider variable values that result from the robot encountering an obstacle at some location with some probability and not encountering the obstacle there. Therefore, a second contribution is a novel method to extend the Bayesian Network formulation of [1] to reason about random variables with different subpopulations.

We also apply our technique to a behavior-based robot program that includes *probabilistic localization* using the Adaptive MonteCarlo Localization algorithm (AMCL)

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

running under ROS [7]. This the first time, to our knowledge, that a formal V&V method has been applied in this way. Verification of this application is challenging because it absolutely requires an environment model, separate from, and interacting with, the behavior-based software. The model has to include the physical location of the robot, the geometry of the map, and the relationship between these and the sensor measurements. A third contribution of the paper are models to include localization in the verification process: a high-level, idealized model and a model for including specific localization (or any probabilistic) software.

An important aspect of our work has been backing up our verification results by extensive, experimental validation. Rather than just presenting the results of verifying mission software for all the missions in this paper, we compare these verification results with performance statistics from experimental validation trials.

The next section reviews the literature in verification of robot software. Section III is a sufficient review of the foundational material from [1] as a basis for the new contributions. Section IV addresses a multi-robot mission that may encounter obstacles, while Section V presents and compares two approaches to verifying a mission with probabilistic localization software. In each case, experimental validation is used to demonstrate that the verification results are consistent with real performance statistics. Section VI summarizes and discusses our novel contributions and future work.

2 Literature

Formal verification can be used as a design tool to determine whether a piece of robot software will function as desired without having to execute the software physically. The field has made significant strides in recent years with the development of model-checking [3] and SMT engines [8]. However, formal verification can at best produce an approximation of robot performance, due to the undecidability of the underlying verification problem. A crucial issue in selecting a verification approach is to understand what aspects of the robot software problem to focus on and how to leverage these to yield efficient automatic verification tools.

Behavior-based robot programming is an important design approach in autonomous robotics because it yields programs that are robust to uncertainty about exactly what environment the robots will face during execution. For this reason, verification of behavior-based robot programs is being addressed by some researchers, e.g., [6] [9] [10], and we also focus on that approach here.

Many robot software verification papers do not include any model of the environment in which the mission is carried out, verifying properties of the software itself such as absence of deadlock or run-time errors [4] [5]. Such an approach might verify that a robot never issues a collision velocity, but not that a robot might roll or be mistakenly pushed into an obstacle – actions that only take place within the environment model. Or it might verify that a bomb-disposal robot has snipped the power wire (the robot's action), but not that the bomb itself has not exploded (a function of the separate state of the bomb).

In some cases, the properties to be verified are used themselves to implicitly express the designer's knowledge (or expectation) of environment dynamics [11]. A simple example of this is assuming that testing for a motor stall is the same as testing for a collision. This informal approach is an error-prone way to capture environment dynamics; a stall might be caused by factors other than a collision.

Some of the most recent verification work does include environment models: The UK EPSRC-funded project on Trustworthy Robotic Assistants proposed representing unstructured environments using the Brahms [12] agent modeling language; however, while this does model environment dynamics, it does not address the crucial issues of motion and sensing uncertainty. These uncertainties can be the difference between success and failure for a critical mission. The latter was identified in [5] as one of the key 'lessons learned' in applying standard formal techniques to robot missions. Fisher et al. [13] address the difficulty of specifying a-priori conditions by verifying the robot's belief rather than its actual behavior. However, the robot's belief may not correspond to what actually happens. In an alternate approach, Guo et al. [14] and Sarid et al. [15] both iteratively produce a correct by construction program as uncertain information becomes known. However, it's not possible with that approach to verify the program in advance.

A common approach to verification is to manually implement the algorithm to be verified in a formal framework. For example, in Proetzsch et al. [16] the robot software to be verified is written in the verification language Quartz. Kim et al. write their robot software to be verified in Esteral [11]. Of course, this reimplementation may not represent the actual software; Published descriptions, even for widely known algorithms, have been shown to contain errors [17]. It also means that verification requires a huge investment of expertise and manpower to rewrite existing robot software into the verification framework [11]. We take a different approach: Mission designers work directly in the *MissionLab* design toolkit, and their software can be automatically translated to PARS [18] for verification – they never have to deal with the formal framework themselves and just use their regular tools for robot mission construction.

Kiekbusch et al. [6] address automatic verification of behavior-based software in their *iB2C* framework. As with our *MissionLab* approach, their software is automatically translated to a verification framework – a set of finite state automata for model-checking. They also provide some environment modelling in the form of *scenarios* which are specific configurations of the environment for testing purposes. However, due to the state explosion problems of their model-checking tool, they can only verify binary behavior activation conditions such as whether an obstacle avoidance behavior is active, rather than the actual motions of the robot in response to the obstacle. They do not represent uncertain information and simply list the scenarios they wish to test against.

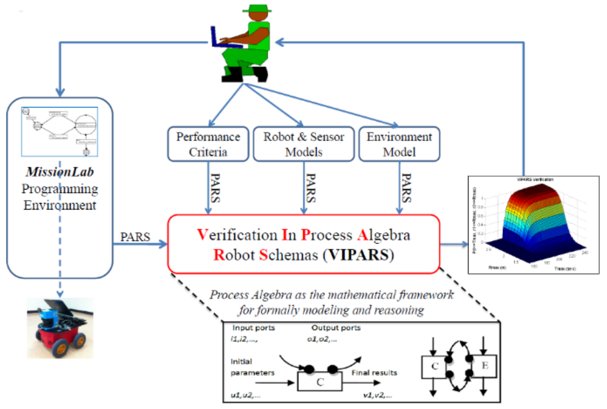
Generally-related work to ours also includes correct-by-construction methods for teams of robots, and verification and validation of planning and scheduling systems. The former focus on automatic synthesis [19], not verification, of a program. In the latter, where a domain model is used to make a plan or schedule to achieve a high-level goal, "experience has shown that most errors are in domain models" [20] – which can only be checked if a separate environment model is included in verification. The work reported in this paper addresses verification using an explicit uncertain environment model.

3 Designing Robot Missions with Verification

This section briefly reviews the material from [1] as a basis for a standalone, self-contained presentation of the new contributions in this paper. The first subsection is an overview of the programming toolkit for designing robot mission software, *MissionLab*, and the way in which automatic verification is added to this toolkit. The next subsection introduces the formal framework PARS (Process Algebra for Robot Schemas) used in verification. The final subsection reviews the verification framework itself, a combination of static analysis and Bayesian networks.

3.1 Mission Design

MissionLab is a usability-tested [21] graphical programming toolkit for robot missions, including graphical editor, mission simulation and execution logging capabilities among others. The mission designer constructs the mission using *MissionLab*, as shown in Figure 1.



MissionLab Programming Environment
Performance Criteria
Robot & Sensor Models
Environment Model
PARS
PARS
PARS
PARS
Verification In Process Algebra Robot Schemas (VIPARS)
Process Algebra as the mathematical framework for formally modeling and reasoning
Input ports
Output ports
Initial parameters
Final results
C
E

Figure 1. *MissionLab*/VIPARS System Architecture.

The VIPARS (*Verification in PARS*) [1] module is designed to work with *MissionLab* and provide a performance verification functionality. VIPARS module inputs include:

- the mission program, as designed in *MissionLab*'s *CfgEdit* graphical interface;
- a set of designer selected library models of the robot, and sensor systems;
- the mission operating environment; and,
- the mission performance criteria.

A mission designer could, for example, construct a single-robot, waypoint mission, indicate that it will take place in a moderately-cluttered warehouse environment, and that it will be performed by a *Pioneer 3-AT* robot equipped with sonar and gyroscope. She could then choose performance criteria that fit the mission (for example, that the robot moves within at least 0.1 meters of each waypoint and finishes all waypoints in under 100 seconds). She can then use VIPARS to verify whether or not the mission will always meet this performance criterion with some given threshold probability.

Prior to the VIPARS module, the mission software is automatically translated to PARS [18] a formal, process-algebra language. The library models of *Pioneer 3-AT*, sonar and gyroscope, and moderately cluttered indoor environment are then combined with the mission software to generate a single PARS system which will be analyzed for the performance guarantee. Our intent is that these robot, sensor and environment models are used, *but not constructed*, by the mission designer; they are built in PARS as probabilistic process models parameterized with robot and sensor calibration data and provided to a designer with the verification module.

VIPARS verifies whether the mission software will achieve the specified performance criteria (typically spatial and temporal constraints) using the selected robot/sensors in the selected operating environment. It also generates predicted performance information that can be used by the designer to either improve the system performance or abort the mission to avert catastrophic failures. The verification component supports an iterative cycle for designing high-performance robot behavior for critical missions.

3.2 PARS

PARS is a process-algebra designed for the purpose of representing robot software, and the robot, sensor and environment models with which the software interacts. The algebraic syntax facilitates developing static analysis algorithms (algorithms that analyze programs without executing them) to identify the interactions between the robot program and its environment. Although PARS was designed for representing robot programs, and in particular robot schemas style, behavior-based programs [22], in fact it shares many characteristics with other process algebras such as CSP [23] and LOTOS [24] and has an operational semantics defined using port automata [25]. As such it could be used to represent any programs, and any robot programming style. However, our more specific results are focused on behavior-based programs because they have a structure that can be leveraged to address efficient verification.

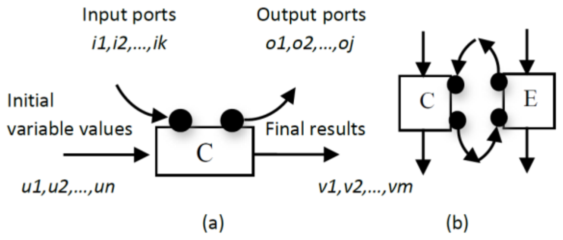


Figure 2: (a) PARS Process; (b) process network (from [1]).

Figure 2 shows the PARS model of a process and process network. A process $\boldsymbol{C}$ (Figure 2(a)) is written as:

$$\boldsymbol{C}\langle \boldsymbol{u_1}, \dots, \boldsymbol{u_n}\rangle(\boldsymbol{i_1}, \dots, \boldsymbol{i_j})(\boldsymbol{o_1}, \dots, \boldsymbol{o_k})\langle \boldsymbol{v_1}, \dots, \boldsymbol{v_m}\rangle \quad (1)$$

where $u_1, \dots, u_n$ are the (finitely many) initial variable values for the variables of the process, $i_1, \dots, i_j$ and $o_1, \dots, o_k$ are input and output port connections, respectively, and $v_1,\ldots,v_m$ are final result values generated by the process.

Processes are either *atomic* or *composite.* A process is defined as a composition of other processes as follows:

```
<processdef>     ::= <process> '=' <processexpr>
<processexpr>    ::= <processeq> '|' <processeq>      |
                     <processeq> '#' <processeq>
<processeq>      ::= <processexpr>';'<processexpr> |
                     '(' <processexpr> ')'          |
                     <processname>
```

Where '|' denotes parallel composition (parallel max), '#' disabling composition (parallel min), and ';' denotes sequential composition, and where <*process*> and <*processname*> are a bolded capital letter or word.

For example, the parallel composition:

$$\mathbf{S} = \mathbf{C}(c1)(c2) \;\; | \; \mathbf{E}(c2)(c1)$$

specifies two parallel processes **C** and **E** as shown in Figure 2(b), with the input and output ports connected correspondingly. The labels *c1* and *c2* are called *port connection labels* and their purpose is to specify the connection map between the ports of the parallel processes.

Each process that terminates can terminate in either a *stop* or an *abort* condition. There is no separate 'choice' operator in PARS. However, a process that evaluates a condition *c* is defined to terminate in a *stop* status if *c* and in *abort* if *not c*. A sequential chain of processes, such as $\mathbf{Eq}\langle x,y\rangle$; **P**, terminates for the first process in the chain that has a termination condition of abort (e.g., if $x \neq y$, **P** is not reached because **Eq** aborts).

Repetitive computation (e.g., loops) is modelled by a tail-recursive (TR) process definitions, written for example:

$$\mathbf{P}\langle x\rangle \; = \mathbf{Q}\langle x\rangle\langle y\rangle \, ; \mathbf{P}\langle y\rangle \tag{2}$$

Eq. (2) defines a process **P** that repeats process **Q** until **Q** aborts, at which point **P** terminates, returning its results. In this example, the process **Q** is the *body* of the TR process, similar to the body of a loop.

A *flow function* $f_P(u_1,u_2,\ldots,u_n)=(v_1,v_2,\ldots,v_m)$ is associated with each **P***,* mapping the values of the variables of **P** at the start to those at the end. The flow-function for atomic processes are specified a-priori, and those for a composite process can be built up from the flow functions of its components, e.g., for $\mathbf{T}\langle x\rangle\langle z\rangle \; = \mathbf{P}\langle x\rangle\langle y\rangle \, ; \mathbf{R}\langle y\rangle\langle z\rangle$ we can say $f_T(x)= f_R \circ f_P(x)$ if **P** does not abort. Static analysis algorithms to calculate flow functions play a key role in VIPARS verification.

3.3 Verification in PARS (VIPARS)

The robot mission software is converted to PARS [18] and combined with the PARS definitions for the robot, sensor and physical environment models (selected by the user) producing a parallel network **Sys** of communicating processes**.** For example, a robot controller **Ctr** with variable r_1, and an environment model[2] **Env** with variable r_2, would be written as:

$$\mathbf{Sys}\langle r_1,r_2\rangle \;\; = \mathbf{Ctr}\langle r_1\rangle(a)(b) \; | \;\; \mathbf{Env}\langle r_2\rangle(b)(a) \tag{3}$$

In the example of eq. (3), the input of **Ctr** (sensor signals) is connected to the output of **Env**, (*a*), and the input of **Env** is connected to the output (control signals) of **Ctr**, (*b*), similar to the process network in Figure 2(b). If eq. (3) were a sequential composition like eq. (2) then we could extract flow functions for the combined interaction of controller and environment and use this function as the basis for verifying all possible executions of the system. However, the addition of port communication complicates the relatively simple definition of flow functions! The flow function associated with a process no longer just depends on the variables of that process, but could depend in a complex way on variables and computations of other parallel processes. To address this, a constraint on the form of parallel compositions is leveraged, namely that all processes are written as tail-recursive (TR) processes. This does not restrict what can be computed but allows us to propose a special static analysis approach to efficiently verifying all possible executions of a behavior-based systems.

In behavior-based robot software, such as that produced by *MissionLab*, sensory information is continually being inspected to determine which behaviors should be activated and how to parameterize them. The software is looking for affordances in the environment that will further the objectives for the mission – as a simple example: moving towards goal locations, but away from obstacles. The intuition is that a behavior-based system has behavioral 'states' each with an associated set of sensory triggered responses.

This is modeled here as a parallel composition of TR processes representing ongoing behaviors or the monitoring of affordances. When a behavior terminates or when an affordance is detected, additional behaviors or affordance monitoring may be added to the parallel composition.

Leveraging the TR structure, *an interleaving theorem*[3] is presented in [1] to convert processes of the form of eq. (3) to a sequential form as shown in eq. (4) below. The intuition here is that the set of TR process bodies can be composed into a single system TR body called the *system period,* shown as the process **Sys'** in eq. (4), and similar to the concept of a hyper-period[4] in process scheduling.

$$\begin{aligned} \mathbf{Sys}\langle r_1,r_2\rangle \;\; &= \mathbf{Ctr}\langle r_1\rangle(a)(b) \; | \;\; \mathbf{Env}\langle r_2\rangle(b)(a) \\ &= \mathbf{Sys'}\langle r_1,r_2\rangle\langle r'_1,r'_2\rangle \;\, ; \mathbf{Sys}\langle r'_1,r'_2\rangle \end{aligned} \tag{4}$$

$$\begin{aligned} f_{Sys}(r_1,r_2) &= (\; f_{Sys,r1}(r_1,r_2), \;\; f_{Sys,r2}(r_1,r_2) \;\;) \\ &= (r'_1, r'_2) \end{aligned} \tag{5}$$

A static analysis algorithm *Sysgen* was developed based on this interleaving theorem to construct the system period, **Sys'** in eq. (4), given the processes, connections and

[2]As a verbal shortcut, we will include the models of the robot, and sensors in the term environment model.

[3] In process algebra, an interleaving theorem relates the sequential and parallel composition operations.

[4] The LCM of all the task periods in a scheduling problem.

communications in **Sys**. This construction reduces the state explosion of all orders of a set of parallel processes to the single *interleaving* of the system period.

Once *Sysgen* analysis is complete, a *system flow function* can be extracted from **Sys'**. In the small example of eqs. (3), (4) above, the function extracted is shown in eq. (5). This is a recurrent function that evaluates the new values for *r1* and *r2* as computed by the interactions between **Ctr** and **Env** in each execution of the system period **Sys'**.

Process variables, such as r_1, r_2 in the example above, can be random or deterministic variables. Typically, mission software variables are deterministic. However, variables in robot, sensor and environment models can be random to represent uncertainty associated with their values. To include both random and deterministic cases, flow functions, which relate variable values at recursion step *i* of **Sys'** to those at *i+1*, can be written as conditional probabilities, e.g.:

$$f_{Sys,r1}\,(r_{1,t}\,,r_{2,t}) = P(r_{1,t+1}\,|r_{1,t}\,,\, r_{2,t}\,) \tag{6}$$

In the final phase of VIPARS processing, extracted flow functions are converted to conditional probabilities. Random variables are represented as multivariate *Mixtures of Gaussians*, and operations on random variables are automatically translated by VIPARS into operations on distributions [26]. These are then the basis of a Dynamic Bayesian Network (DBN) [27] used to carry out forward propagation of probability distributions, to determine whether the combination of controller and environment will meet a performance specification.

Although [1] discusses more complicated performance guarantees, we will typically restrict our attention to the guarantee that a mission will achieve some criterion on environment variables (usually a spatial accuracy for a waypoint goal and/or a temporal requirement for achieving the mission) with probability greater than a threshold before a time-limit. We demonstrated that this approach is fast and accurate when validated against physical executions (most recently [28]).

4 Multirobot Mission with Uncertain Obstacles

Bounding overwatch is a military movement tactic used by units of infantry to advance forward when crossing dangerous areas [29]. In the first mission we will address, a team of two robots will use this strategy to move stealthily inside a building to search for biohazards which may be guarded by hostile forces and in which they may encounter obstacles along their route.

Figure 3 shows the *bounding overwatch* mission where two robots coordinate their movements in a "leapfrogging" manner while advancing toward a biohazard. Robot2 begins by bounding toward ***O1***, the first ***O****verwatch* position. When it reaches ***O1***, a *"Cleared"* message is sent to Robot1 indicating that it is safe to proceed. Robot1 then bounds to ***O2*** and sends the *"Cleared"* message to Robot2, and so on. The mission ends with Robot2 at ***O7***, near the biohazard. The operating environment of this mission includes some obstacles whose existence or exact locations are not known with certainty in advance; if they are present, the obstacles will be within the locations illustrated with dashed circles shown in Figure 3. This lack of a-priori certainty about the environment geometry is a challenge for verification in efficiently representing and checking all the potential obstacle-related motions of both the robots.

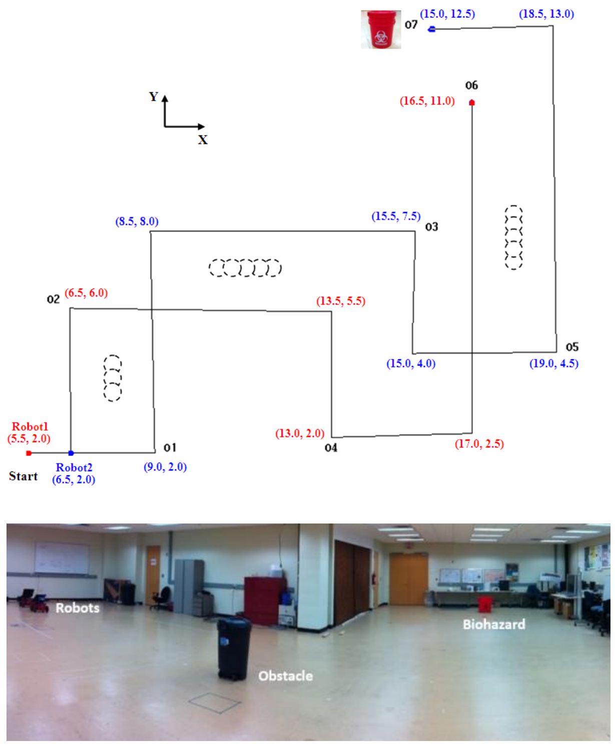


Figure 3. Bounding Overwatch with Two Robots: Map (top), Operating Environment (bottom).

4.1 *Bounding Overwatch Mission*

The behaviors of Robot1 and Robot2 are specified graphically in *MissionLab* as behavioral finite state automata (FSAs). Each behavioral FSA consists of the following behaviors:

- *GoToGuarded:* Move to a waypoint while avoiding obstacles;
- *NotifiedRobots:* Send a "*Cleared"* message to the other team members;
- *Spin*: rotate the robot;
- *Stop:* mission concluded;
- *AtGoal:* sensory trigger for arrival at location;
- *HasTurned:* sensory trigger for arrival at orientation*;*
- *Notified:* sensory trigger for "*Cleared"* message;
- *MessageSent*: sensory trigger for message sent.

The behavioral FSA of triggers and behaviors for Robot1 is shown in Figure 4 and that for Robot2 is similar.

The behavioral FSA is translated to a *MissionLab* internal language called CNL [2] and a translator from CNL to PARS [18] produces a process model of the program which includes detailed specifics of all the behaviors and triggers. The following performance criteria are used to evaluate this mission performance:

$$Success = (r_1 \le R_{max}) \text{ AND } (r_2 \le R_{max}) \text{ AND } (t \le T_{max}) \qquad (7)$$

Where r_1 and r_2 are Robot1's and Robot2's relative distances to their respective goal and t is the mission completion time, where R_{max} is the success radius, and T_{max} is the maximum allowable mission time. The *bounding overwatch* mission is only considered successful when both robots are within R_{max} radius of their respective goal locations and when they complete the mission under T_{max} seconds.

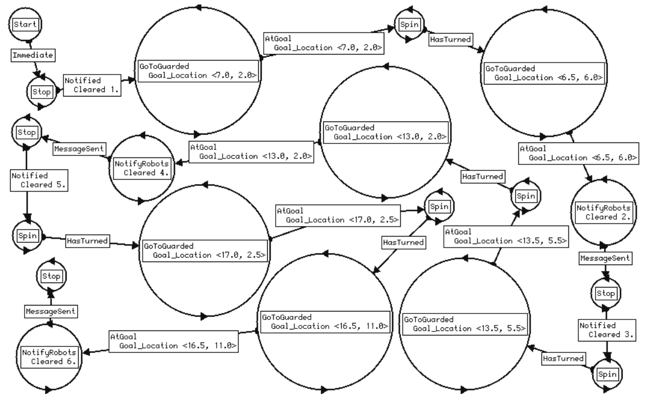

Start
Immediate
Stop
Notified Cleared 1.
GoToGuarded Goal_Location <7.0, 2.0>
AtGoal Goal_Location <7.0, 2.0>
Spin
HasTurned
GoToGuarded Goal_Location <6.5, 6.0>
Stop
MessageSent
NotifyRobots Cleared 4.
AtGoal Goal_Location <13.0, 2.0>
GoToGuarded Goal_Location <13.0, 2.0>
AtGoal Goal_Location <6.5, 6.0>
Notified Cleared 5.
HasTurned
Spin
NotifyRobots Cleared 2.
AtGoal Goal_Location <17.0, 2.5>
Spin
Spin
HasTurned
GoToGuarded Goal_Location <17.0, 2.5>
HasTurned
AtGoal Goal_Location <13.5, 5.5>
MessageSent
Stop
Stop
MessageSent
GoToGuarded Goal_Location <16.5, 11.0>
GoToGuarded Goal_Location <13.5, 5.5>
Notified Cleared 3.
NotifyRobots Cleared 6.
AtGoal Goal_Location <16.5, 11.0>
HasTurned
Spin

Figure 4. Behavioral FSA for Robot1.

4.2 Robot Motion Model

In [1] we presented a PARS robot process model **Robot** with motion and position sensing uncertainty.

$$\begin{aligned}\mathbf{Robot}\langle p, a, s\rangle = {} & (\,\mathbf{Delay}\langle \tau\rangle \;\#\; \mathbf{Odo}\langle \mathrm{p}\rangle \# \;\mathbf{At}\langle \mathrm{r1, p}\rangle\,)\;;\\ & (\,\mathbf{In}\langle \mathrm{d}\rangle\langle \mathrm{h}\rangle \;\#\; \mathbf{In}\langle \mathrm{v}\rangle\langle \mathrm{s}\rangle\,)\;;\\ & (\mathbf{Ran}\langle \Theta_h\rangle\langle z\rangle \;|\; \mathbf{Ran}\langle \Theta_v\rangle\langle w\rangle\;)\;;\\ & \mathbf{Robot}\langle \mathrm{p+(h+z)*(s+w)*}\,\Delta t,\; \mathrm{a},\; \mathrm{s}\,\rangle.\\ \mathbf{Odo}\langle r\rangle \quad = {} & \mathbf{Ran}\langle \Phi\rangle\langle \mathrm{e}\rangle\;;\mathbf{Out}\langle \mathrm{p, r+e}\rangle\;;\;\mathbf{Odo}\langle \mathrm{r}\rangle.\end{aligned}$$

Robot accepts a unit vector heading input on port d or a speed in the direction of the heading on port v. The process **At**⟨r1, p⟩ represents robot $r1$ at location p. The process **Odo** (Odometry sensor) makes position information (with noise $e \sim \Phi$) available in a loop on port p until terminated by **Delay**. The new position of the robot is calculated as the old position p incremented by a speed s with added noise w in the direction of the commanded heading h with added noise z. The odometer position is the actual position with added noise e. The actuator and odometer noise (the variables z, w, and e) are characterized by the distributions for speed, heading and sensor noise, e.g., $\Theta_h = N(\mu_h,\sigma_h)$, $\Theta_v = N(\mu_v,\sigma_v)$, and $\Phi = N(\mu_m,\sigma_m)$. The flow function for the position variable of the robot model, with operations on random variables translated to operations on distributions, is

$$p_{t+\Delta t} = p_t \cdot (\,\Theta_h \cdot h_{t+\Delta t}\,) \cdot (\,s_t \cdot \Theta_v\,) * \Delta t,$$

where '$\cdot$' denotes convolution.

The robot model used in this paper follows this same structure but is more detailed in its representation of the motion uncertainty. The new robot position distribution $p_{t+\Delta t}$ is calculated as the old position distribution p_t convolved with $s_{t+\Delta t}\, h_{t+\Delta t}\, \Delta t$ - a nominal change at speed s_t with heading unit vector h_t for time Δt and a motion uncertainty term. The latter is a convolution of a translational TX, rotational TR and skitter TS uncertainty component:

$$TX(s_{t+\Delta t}, h_{t+\Delta t}) = X \cdot \frac{s_{t+\Delta t}}{s^k} \cdot H(s_{t+\Delta t} - 1) \cdot R(h_{t+\Delta t}) \cdot \Delta t$$

$$\begin{aligned}TR(s_{t+\Delta t}, h_t, h_{t+\Delta t}) &= U \cdot |h_t - h_{t+\Delta t}| \cdot\\ &\qquad (2H(\theta_t - \theta_{t+\Delta t}) - 1) \cdot R(h_{t+\Delta t}) \cdot \Delta t\end{aligned}$$

$$\begin{aligned}TS(s_{t+\Delta t}, h_t, h_{t+\Delta t}) &= W \cdot |h_t - h_{t+\Delta t}| \cdot H(1 - s_{t+\Delta t}) \cdot\\ &\qquad (2H(\theta_t - \theta_{t+\Delta t}) - 1) \cdot R(h_{t+\Delta t}) \cdot \Delta t\end{aligned}$$

where

- X, U and W are robot specific bivariate normal distributions calibrated by multiple measurements over a range of distances and angles for each robot at calibration speed s^k.
- $R(h)$ is an operator that rotates a bivariate distribution by the unit vector h.
- $H(x)$ is the Heaviside step function.
- θ_t is the angle of heading unit vector h_t.

In summary, the flow function for the position distribution $p_{t+\Delta t}$ is now:

$$p_{t+\Delta t} = p_t \cdot (s_t h_{t+\Delta t} \Delta t) \cdot (TX \cdot TR \cdot TS) \qquad (8)$$

4.3 Multiple Robots

This mission uses two robots, and the two robots interact *directly* to synchronize their motion (Figure 4) and *indirectly* as physical obstacles to each other. Multiple robots are added to the environment model by creating extra instances of the **Robot** process, each with unique X, U and W calibration distributions measured by calibration experiments on a physical robot. Each robot process has its own input and output ports through which the mission software can communicate. No extension to the process algebra model was required to handle the multiple robot case.

The direct communication between the robots is modelled for verification by a simple, centralized communication structure. The *Notify* and *MessageSent* behaviors map to port read and write commands from the PARS translation of Figure 4 (and its equivalent for Robot2) to a single communication process **Comm**. No message transmission latency or error was modelled for this example.

4.4 Uncertain Geometry Model

Since the geometry of the environment is not completely known in advance, we construct a probabilistic model that includes whatever a-priori information there is. One way to generate such a model is as shown in Figure 5: Several spatial locations along the mission are annotated a-priori as being potential obstacles; this is the approach we will take in this mission. Another approach would be to use the map output from probabilistic mapping software that has been used to measure the environment – we will take that approach in a later section. Of course, both approaches could be combined.

For this mission, the physical environment is modeled as a collection of isotropic bivariate Gaussian mixtures: Figure 5(a) shows a mixture of 8 members modelling a rectangular 2D obstacle. Figure 5(b) shows the model with 16 members.

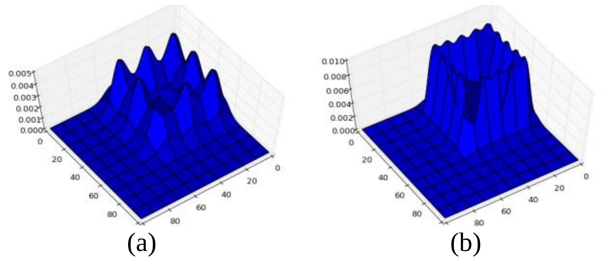


Figure 5. Modelling geometry with bivariate Gaussian mixtures.

The *GoToGuarded* behavior in *MisssionLab* is translated to the process network shown in (9) which implements its behavior:

$$\begin{aligned}&\textbf{Coop}\,\langle 1,1,1\rangle\,(vg, vo, vn)(v) \quad |\\&\textbf{Move\_to}\,\langle P0,G3\rangle\,(pR)(vg) \quad |\\&\textbf{Noise}\,\langle ns\rangle\,(pR)(vn) \quad |\\&\textbf{Avoid\_Obstacles}\,\langle r\rangle\,(pR,obR)(vo)\end{aligned} \tag{9}$$

The **Avoid_Obstacles** process inputs robot position (through connection *pR*) and any sensed obstacles (on *obR*) and generates a potential-field based avoidance velocity output (*vo*) [22]. **Move_to** generates a velocity towards a waypoint *G3* (*vg*) and **Noise** generates a small velocity perturbation to escape potential minima (*vn*). The **Coop** process combines all three vectors into a single command velocity (*v*) with equal weights (1,1,1).

In execution, the input and output of these processes correspond to the connections of *GoToGuarded* with the real robot and its sensors, and through these, with the actual execution environment. However, in verification, this information is provided instead by robot, sensor and physical environment models selected by the user. For this mission, these are shown below:

$$\begin{aligned}&\textbf{Robot}\langle P0, \Delta t, \varphi\rangle\,(v)(pR) \quad |\\&\textbf{Sensors}\langle S0, sr, sn\rangle\,(pR, pE)(s) \quad |\\&\textbf{Geometry}\langle E\rangle\,(pR, pR2)(pE)\end{aligned} \tag{10}$$

The **Robot** process takes a velocity command (on *v*) and generates a new position distribution (on *pR*) according to (8) and where $p_t, v_t \sim MG(M_p)$ are modeled as mixtures of bivariate Gaussians representing the 2-D location and velocity of the robot. The **Sensors** process calculates what obstacle locations will be sensed by the robot, implemented as follows:

$$\begin{aligned}&\textbf{Sensors}\langle S0,sr,sn\rangle\,(pR,pE)(obR) \;=\\&\quad \textbf{In}\langle pR\rangle\langle p\rangle\,;\,\textbf{In}\langle pE\rangle\langle e\rangle\,;\\&\quad (\,\textbf{Gtr}\langle\, d(p,e), sr\rangle\langle p1\rangle\,;\,\textbf{Out}\langle obR, p1\rangle \;|\\&\quad\;\; \textbf{Lte}\langle\, d(p,e), sr\rangle\langle p2\rangle\,;\,\textbf{Out}\langle obR, sn{+}p2\,\rangle\,);\\&\quad \textbf{Sensors}\langle S0, sr, sn\rangle\,.\end{aligned} \tag{11}$$

The robot position (on *p*) and geometry (on *e*) are inputs from whatever **Sensors** has been connected to - in this case, the **Robot** process and the **Geometry** process. **Geometry** continually adds the latest position distributions for both robots to the static geometry distribution (obstacles) and transmits this to the **Sensors** process (*e*). This approach handle the indirect interaction between robots, and it generalizes linearly to any number of robots.

The distance function *d(p,E)* calculates what portion of the environment is within the sensor range (*sr)*. The procedure for determining potential collisions and sensor feedback involves computing the Bhattacharyya Coefficient [30] between robot position and the geometry distribution. This coefficient measures the amount of overlap between two multivariate normal distributions as follows:

$$\begin{aligned}&\boldsymbol{BC}\big(\boldsymbol{N}(\boldsymbol{\mu_0},\boldsymbol{\Sigma_0}),\boldsymbol{N}(\boldsymbol{\mu_1},\boldsymbol{\Sigma_1})\big)\\&= \mathbf{exp}\left(-\frac{\mathbf{1}}{\mathbf{8}}(\boldsymbol{\mu_0}-\boldsymbol{\mu_1})^{\boldsymbol{T}}\boldsymbol{\Sigma}^{-\mathbf{1}}(\boldsymbol{\mu_0}-\boldsymbol{\mu_1})\right)\sqrt{\frac{\sqrt{|\boldsymbol{\Sigma_0}||\boldsymbol{\Sigma_1}|}}{|\boldsymbol{\Sigma}|}}\\&\boldsymbol{where}\ \boldsymbol{\Sigma} = \frac{|\boldsymbol{\Sigma_0}||\boldsymbol{\Sigma_1}|}{\mathbf{2}}\end{aligned} \tag{12}$$

d(p,e) generates a bivariate distribution with members corresponding to the joint probabilities between the members of the p and e variables. The result of sensing (*obR*) is this distribution (convolved with a sensor noise distribution (*sn*)).

4.5 *Conlicting Hypothesis Histories*

The flow functions automatically extracted by VIPARS from the *GoToGuarded* network (9) connected to the environment model (10) include the effects of condition processes (such as **Gtr** and **Lte**) and can be written in terms of the Heaviside step function *H(.)* and unit vector *u(.)*. Operations on variables (e.g., addition) are translated to equivalent operations on distributions (e.g., convolution). The following are among the flow functions extracted and just come from the definition of the *GoToGuarded* behavior:

$$\begin{aligned}&f_{vo}(s, p) = r - H(r - so_t)so_t,\\&f_{vg}(p, g) = u(\,p_t - g\,)so_{max}\\&f_v(v_o,v_g, v_n) = v_o\cdot v_g\cdot\; v_n\end{aligned} \tag{13}$$

The obstacle velocity (vo in (9)) is specified by f_{vo} as linearly proportional to the distance to the obstacle $r - so_t$ but at most *r* if there are obstacles seen. The goal velocity f_{vf} is a fixed velocity s_{max} in the direction of the goal $u(\,p_t - g\,)$. (In fact, there is a ramp-down to the goal, omitted here for simplicity.) The final velocity is just the convolution of the noise, obstacle and goal velocities (the result of **Coop** in (9)).

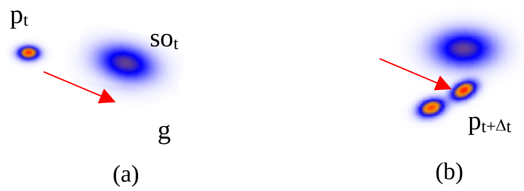


Figure 6. Example of Obstacle Avoidance.

Consider the example shown in Figure 6: At some time *t*, the position (p_t, a single member distribution) is close enough to a sensed obstacle so_t that an obstacle repulsive velocity (*vo*) is generated in addition to the velocity towards the goal (*vg*) (Figure 6(a)). The portion of the position distribution that resulted in no obstacle detection (*p1* in (11)) should be convolved with just a forward velocity; the portion that had obstacle detection (*p2*) should be convolved with both forward and repulsion (Figure 6(b)), capturing both potential outcomes.

In fact, however, there is *insufficient information* in the random variable model used by VIPARS to correctly represent this situation. During forward propagation of probability by the DBN, the information of sensor returns where collisions are predicted *becomes separated* from the information about which robot locations generated those returns. Informally: the p_t mixture could be considered as a weighted collection of (Normal distribution) hypotheses for the robot position. The sensory data is generated from this list, but the correspondence between a sensory data mixture member, which originates from so_t, and the hypotheses in p_t that generated the member can be complicated:

1) If the geometry *g* is a multimodal distribution (almost certainly would be), then each member of p_t will generate at least many modes within so_t due to (11).
2) The conditional nature of f_{vo} (i.e., the step-function) means that not every member of so_t generates a repulsive velocity (e.g., because it's too far away).
3) The final, convolution for f_v in (13) will apply goal and repulsion velocities to *all* position modes, not just the ones as shown in Figure 6.

4.6 Colored Mixture of Gaussians (CMG).

The solution to this dilemma is to allow subpopulations of the location variable to be tagged, and for this tag to be propagated to the sensing distribution, so that it becomes clear how the sensing relates to position. The mixture representation for random variable is extended as follows.

Definition. A *colored mixture of Gaussians* (CMG) is a mixture of Gaussians distribution in which each mixture member (mode) is tagged with a color label. If $a \sim CMG(CM)$, for $CM=\{(\mu_i, \Sigma_i, w_i, c_i) \mid i \in 1\ldots m\}$ the set of the mixture parameters (means, variances weights, and colors respectively), then a_i will refer to $N(\mu_i, \Sigma_i)$, $w(a_i)=w_i$ and $c(a_i)=c_i$. The mixture size is written $|\ a\ | = m$. A CMG is evaluated at a point *x* in the usual way as $CMG(x\ ;\ CM)$:

$$\boldsymbol{CMG}(\boldsymbol{x}; \{(\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i}, \boldsymbol{w_i}, \boldsymbol{c_i}) \mid \boldsymbol{i} \in \boldsymbol{1}, \ldots, \boldsymbol{m}\}) = \sum_{i=1}^{m} \boldsymbol{w_i}\, \boldsymbol{N}(\boldsymbol{x}; \boldsymbol{\mu_i}, \boldsymbol{\Sigma_i}), \quad \sum_{i=1}^{m} \boldsymbol{w_i} = \boldsymbol{1} \tag{14}$$

The color tags allow related subpopulations of the CMG to be similarly transformed. Operations on random variable can now be converted to *color-respecting* operations. A color-respecting convolution operation in f_v of (13) can be defined:

Definition. The color respecting convolution $r = p \otimes q$, r, p, $q \sim CMG$ is defined using the notation of the CMG definition as: $r_i = p_j * q_k \Leftrightarrow c(p_j) = c(q_k)$ with weights $w(r_i)$ adjusted accordingly.

As an example, let p_t have two members, p^1 and p^2, and if there are two members of the geometry distribution, o^1 and o^2, then so_t will have four members, two with $c(p^1)$ and two with $c(p^2)$ transformed by the (unimodal) sensor uncertainty distribution (sn). The *color respecting convolution* operation in f_v (13) will result in four velocity members: one for v_g and one for the sum of v_g for p_1 plus the sum of the two v_o with *color*(p_1), and two similarly for p_2. If the step function in f_{vo} trims members from so_t, the members of v_o and v_g can still be correctly matched by color.

With this modification to the random variable framework of VIPARS – namely, the addition of color tags to the multivariate mixture model, and the extension of random variable operations (not just convolution) to respect color – the uncertain geometry model can be used to verify multirobot missions that include obstacle avoidance strategies. The next section presents evidence for this.

4.7 Verification and Validation

The Overwatch mission presented in Subsection A is verified using the modified CMG filtering and the verification results experimentally validated in this section. In the interest of providing more than just a binary verification result, VIPARS produces a graph of the probability of mission success versus time (Time Criterion graph) and graph of the probability of final positional accuracy (Spatial Criterion Graph).

4.7.1 Mission Validation

Each validation run consists of real robots carrying out the *Overwatch* mission. The operating environment of the mission is an indoor lab environment with tile floor. The biohazard is represented by a red bucket marked with the biohazard symbol. The obstacles are green trashcans with radii of approximately 0.25m. The dashed circles in Figure 3 represent the potential locations of the obstacles. The number of obstacles (i.e., 1 to 3) and their locations are varied for each validation run, to reflect the uncertainty of their presence in the environment. At the end of each validation run, the following measurements relating to the performance criteria R_{max} and T_{max} are recorded:

1. $\boldsymbol{r_1}$ – Robot1's relative distance to its goal location;
2. $\boldsymbol{r_2}$ – Robot2's relative distance to its goal location;
3. ***t*** – Mission completion time.

The complete validation experiment consists of 100 trials (calculated to cover all obstacle locations uniformly). The result of the validation experiment is compared to the verification result in the following subsection. These two results were generated without knowledge of each other and only compared after each was completed.

4.7.2 Comparison of verification and validation results

Besides generating accurate results, how to present verification results (i.e., performance guarantees) to the mission designer is also an important research question. We present a preliminary representation that consists of two steps: 1) define performance guarantee as the probability of success (i.e., the probability of meeting a performance criterion) and 2) divide the success probability into confidence regions.

Figure 7 shows the verification and validation spatial criteria for this mission as the probability that both robots are within R_{max} radius of their respective goal locations $P(r_1 \le R_{max}, r_2 \le R_{max})$ versus R_{max}. The graph has three regions based on VIPARS verification: 1) *High Confidence (Unsuccessful)*, *2) Uncertain*, and 3) *High Confidence (Successful)*. The *High Confidence (Unsuccessful)* region is where VIPARS predicts a zero probability of success, informing the operator that she should abort the mission or modify mission parameters (e.g., use different robots) if the verification result is in this region. The *High Confidence (Successful)* region is where VIPARS guarantees success with probability 1.0.

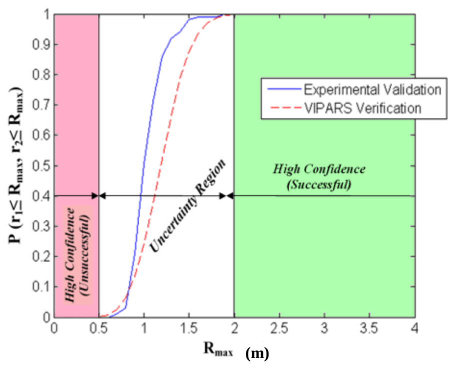


Figure 7. Verification vs. Validation of Spatial Criterion $P(r_1 \le R_{max}, r_2 \le R_{max})$.

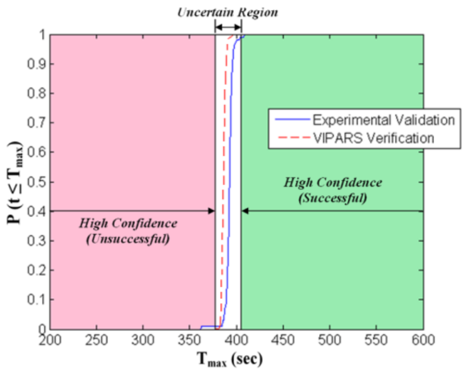
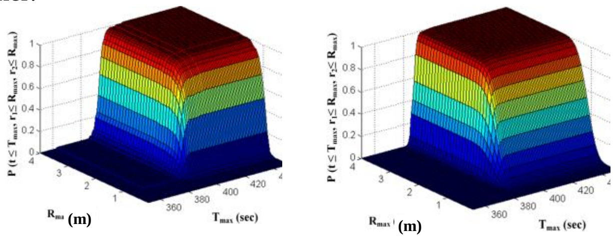


Figure 8: Verification vs. Validation of Time Criterion $P(t \le T_{max})$.

The mission operator has a special interest in this region since she expects the robots would *get it right the first time* for mission requirements (e.g., R_{max}) within this region. The region between *High Confidence (Unsuccessful)* and *High Confidence (Successful)* is defined as the *Uncertain* region, which corresponds to the region where the values of the VIPARS's mission success probability are between 0 and 1.0. In this region, the robots are not guaranteed to get it right the first time. Although both verification and validation curves are shown on Figures 7 and 8, <u>*only*</u> the verification curve is used to define these regions.

Figure 8 shows the verification and validation (performed over 100 trials as described) for the time criterion as a graph of the probability that the mission completes by *t*, $P(t \le T_{max})$ versus *t*. The graph is again divided into the three confidence regions. We observed that most of the discrepancies between verification and validation are within the *Uncertain* region. The region is relatively small, and both validation and verification curves rise sharply, indicating that the boundary of 0% successful and 100% successful is relatively sharp.

We also observed some discrepancies outside the *Uncertain* region, near its boundaries. Ideally, all the errors should be within the *Uncertain* region. However, the errors between the verification and validation success probabilities outside the *Uncertain* region are actually ≤0.01 (i.e., within ~1.01% error). At the boundary between *Uncertain* and *High Confidence* regions, VIPARS predicts a success probability of 1.0 while the actual experimental validation had a success probability of 0.9901, which resulted in a verification error of 0.0099. So, it is still justified to have a high confidence of mission success in the uncertain region since the experimental validation has a success probability of 0.99 and higher.



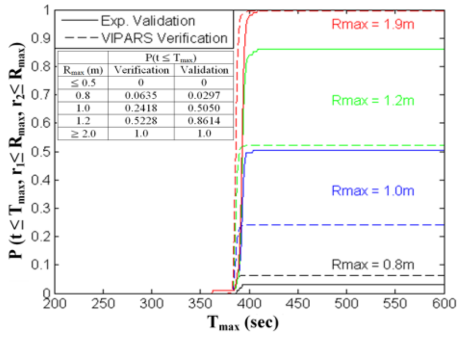
a) Experimental Validation b) VIPARS Verification

Figure 9: Validation (a) and Verification (b) of Overall Mission Success $P(r_1 \le R_{max}, r_2 \le R_{max}, t \le T_{max})$.

We have examined individual performance criterion separately thus far. However, the overall mission success was defined in terms of both spatial and time criteria. Figure 9 shows the verification and validation of the performance guarantee for the overall mission success, $P(r_1 \le R_{max}, r_2 \le R_{max}, t \le T_{max})$, the probability that the bounding overwatch mission is completed under the time limit T_{max} and both robots are within R_{max} radius of their respective goal position. The effect of different combinations of performance criteria values is further examined in Figures 10-11. Figure 10 shows the verification and validation of the time criterion, $P(t \le T_{max})$, at various fixed values of the spatial criterion, R_{max}. We observed that R_{max} in both high confidence regions (i.e., $R_{max} \le$ 0.5m and $R_{max} \ge$2.0m, Figure 6) has no effect on $P(t \le T_{max})$. However, R_{max} in the *Uncertain* region (e.g., R_{max}= 0.8m, 1.0m, 1.2m) has significant impact on $P(t \le T_{max})$. Specifically, $P(t \le T_{max})$ plateaus at different probability values for different R_{max}'s in the *Uncertain* region. For instance, for R_{max} of 1.2m, $P(t \le T_{max})$ plateaus at 0.5228, which is the value of $P(r_1 \le 1.2, r_2 \le 1.2)$ for the spatial criterion in Figure 7.

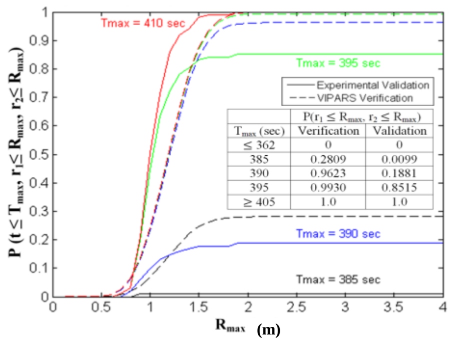
There is a significant discrepancy between verification and validation of $P(t \le T_{max})$ when R_{max}'s are in the *Uncertain* region (max 400 mm). Similar observations are made in Figure 11 for $P(r_1 \le R_{max}, r_2 \le R_{max})$ at various values of the time criterion, T_{max}. These observations reinforced our view that performance criteria within the *Uncertain* region should be avoided, or be moved into the *High Confidence (Successful)* region by modifying mission parameters such as modifying the mission velocity limits or use different robots or sensors.

For this paper, no attempt was made to manually or automatically modify mission parameters to improve mission performance based on verification results; the focus here was on the initial comparison of verification and validation.



	$P(t \leq T_{max})$	
R_{max} (m)	Verification	Validation
≤ 0.5	0	0
0.8	0.0635	0.0297
1.0	0.2418	0.5050
1.2	0.5228	0.8614
≥ 2.0	1.0	1.0

Figure 10: Verification and Validation of Time Criterion $P(t \leq T_{max})$ at various R_{max}.



	$P(r_1 \leq R_{max}, r_2 \leq R_{max})$	
T_{max} (sec)	Verification	Validation
≤ 362	0	0
385	0.2809	0.0099
390	0.9623	0.1881
395	0.9930	0.8515
≥ 405	1.0	1.0

Figure 11: Verification and Validation of Spatial Criterion $P(r_1 \leq R_{max}, r_2 \leq R_{max})$ at various T_{max}.

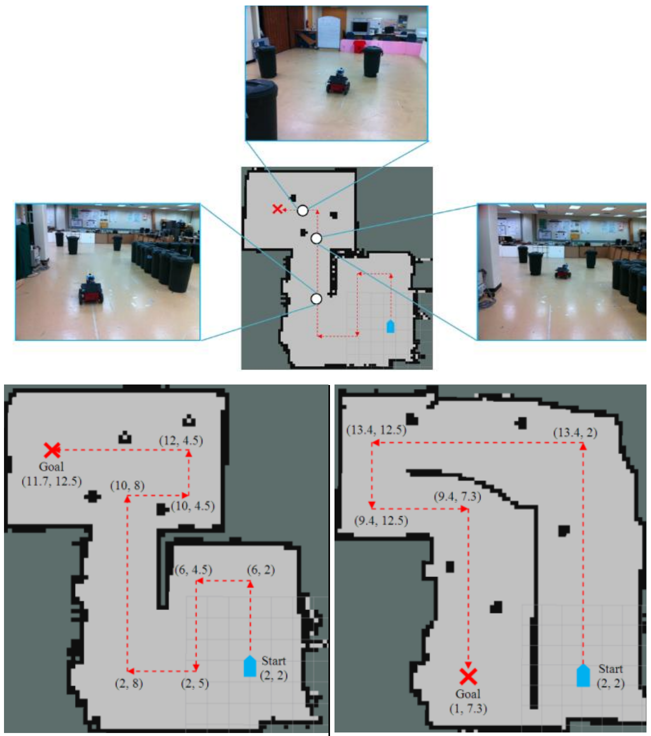
5 VERIFYING MISSIONS WITH LOCALIZATION

To assess the effectiveness of verification in providing performance guarantees for probabilistic robot behaviors, we analyze two waypoint missions, where in each a robot is tasked to navigate through a series of waypoints toward a goal with behaviors that are based on *probabilistic localization algorithms*. VIPARS is used to investigate two approaches to modeling localization and the results compared to experimental validation.

5.1 *Localization Missions-A and B*

Both missions proceed with a robot starting at (2, 2) in Figure 12 and following a series of waypoints to the goal locations at (11.7, 12.5) and (1.0, 7.3) respectively for *Mission-A* and *Mission-B* (respectively). The behavior of the robot for *Mission-B* (Figure 12(a)) is shown in Figure 13, which was created in *MissionLab* in the form of a behavioral FSA. The robot FSA consists of a series of *GoToGuarded* and *Spin* behaviors, whose transitions are prompted by *AtGoal* and *HasTurned* triggers. The behavioral FSA for *Mission-A* is like the one shown in Figure 13, and is omitted for brevity.

The perceptual schemas of *MoveToGuarded* and *AvoidObstacles*, two of the constituent primitive behaviors of the high-level *GoToGuarded* behavior, are augmented with a SLAM-based spatial map [7]. The *MoveToGuarded* primitive behavior drives the robot to a specified goal.



a. Waypoints Mission A

b. Waypoints Mission B

Figure 12: (top) Example Operating Environment Images for Localization missions; (bottom a, b) Two Waypoint Missions for Verification and Validation.

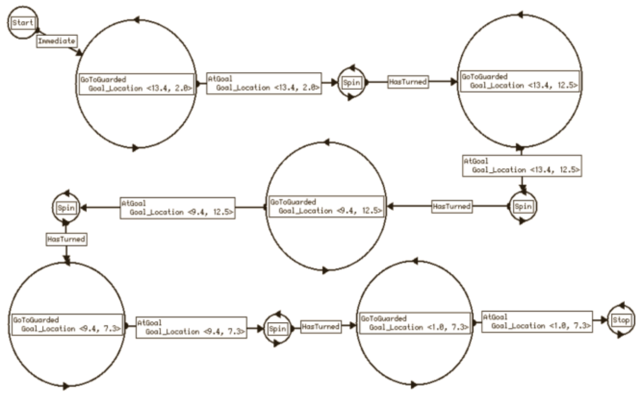

Start
Immediate
GoToGuarded Goal_Location <13.4, 2.0>
AtGoal Goal_Location <13.4, 2.0>
Spin
HasTurned
GoToGuarded Goal_Location <13.4, 12.5>
AtGoal Goal_Location <13.4, 12.5>
Spin
HasTurned
GoToGuarded Goal_Location <9.4, 12.5>
AtGoal Goal_Location <9.4, 12.5>
Spin
HasTurned
GoToGuarded Goal_Location <9.4, 7.3>
AtGoal Goal_Location <9.4, 7.3>
Spin
HasTurned
GoToGuarded Goal_Location <1.0, 7.3>
AtGoal Goal_Location <1.0, 7.3>
Stop

Figure 13: Behavioral FSA for *Mission-B.*

Instead of using odometry for localization, the perceptual schema of *MoveToGuarded* is replaced by an Adaptive Monte Carlo Localization (AMCL) algorithm [31]. This probabilistic localization algorithm takes the robot odometry and an a-priori acquired map as inputs, and outputs an estimated pose of the robot along with a covariance matrix representing the uncertainty of the estimated pose. Furthermore, the *AvoidObstacles* behavior uses the spatial map, instead of using direct sensory reading from the laser scanner, to generate repulsion vectors. The perceptual schema of the

AvoidObstacles is modified to turn the spatial map into pseudo laser scans of the environment through beam tracing within the occupancy map. As a result, the *GoToGuarded* behavior utilizes perceptual information (i.e., robot pose and obstacles) generated by probabilistic algorithms to generate motor response while navigating through the waypoints.

The performance criteria for both missions are similar:

$$Success = (r \leq R_{max})\ AND\ (t \leq T_{max}) \tag{15}$$

Where R_{max} is maximum radius of spatial deviation allowed from the goal and T_{max} is the maximum allowable mission completion time, and where r is the robot's relative distance to its goal location and t is the time the robot to finish a mission.

5.2 *Localization Mission System Process*

The input to VIPARS is the system process composed of the behavior FSA from *MissionLab* converted to PARS and combined with the PARS models for the robot, sensors and environment. The system process **Sys** for the localization mission is shown in eq. (16).

$$\begin{aligned}\textbf{Sys} = (\ & \textbf{Mission}\,(\text{clp, clh, cl})(\text{cv}) \mid \\ & \textbf{Map}\langle\text{sysmap}\rangle()(\text{cm}) \mid \\ & \textbf{Localization}\langle\text{D0}\rangle(\text{cp,co,ch,cl,cm})(\text{clp,clh}) \mid \\ & \textbf{MB\_Laser}\langle\text{ms, mo ,lo}\rangle(\text{cm,cp,ch})(\text{cl})\ ) \mid \\ & \textbf{Robot}\langle\text{P0,H0}\rangle(\text{cv})(\text{cp, ch, co})\ .\end{aligned} \tag{16}$$

The **Mission** process is the translation of the waypoint mission and is fundamentally similar to all prior waypoint missions we have verified and validated. It has inputs *clp* (position), *clh* (heading) and *cl* (laser readings); and output *cv* (velocity). **Robot** is the environment model, capturing the motion and odometry error and interactions with obstacles, as before. PO, HO are initial position and heading, inputs *cv* (velocity) and outputs *cp, ch* (odometry position and heading) and *co* (real position distribution, i.e., without sensing noise – only used for performance estimation and high-level localization model).

However, there are three new processes: In the behavior-based localization approach [7], the obstacle avoidance sensor gets its information from the map, rather than directly from measuring sensory input. **Map** makes mapping information (from the a-priori generated *sysmap*) available on its output *cm*; **MB_Laser** uses the map to generate map-based laser data on its output *cl.* **Localization** implements a localization method using the map *cm*, laser *cl*, and robot *cp, co, ch* inputs. *D0* is the initial position uncertainty. The output of **Localization,** *clp*, is the localized position (and heading *clh*) used by the **Mission** process.

5.3 *Map Representation*

A key difference between this localization mission and our prior missions including bounding overwatch is the map and the role it plays in the obstacle avoidance behavior and in localization. The **Map** process in eq. (16) contains a map data structure. Recall that variables in a PARS process definition can be random variables represented as colored mixtures of Gaussians distributions (*CMG*).

Map information – the locations and geometry of obstacles, walls and other physical aspects of the mission environment – can be directly represented using this model. The interactions of the map with the robot and map-based sensor is analyzed in VIPARS by measuring the overlap between random variable distributions, eq. (12). The advantage of this approach to representing physical geometry is that there is no restriction on the spatial location or extent of obstacles, and finer precision of modeling can be obtained at the cost of adding more mixture members (Figure 14).

Definition. An *indexed mixture of Gaussians* is a mixture of Gaussians distribution $a \sim CMG(CM)$ together with an index set I. The mixture is restricted as follows:

- $a[x] \equiv a_i$ where $\mu(a_i) = x \in I,\ i \in 1 \ldots m$.
- $\mu(a_i) \in I$, *for all* $i \in 1 \ldots m$; *a* only contains members indexed by $\underline{I}$.
- For any $x \in I$, $|\{a[x]\}| \leq 1$; $\underline{a}$ has at most one member for each index.

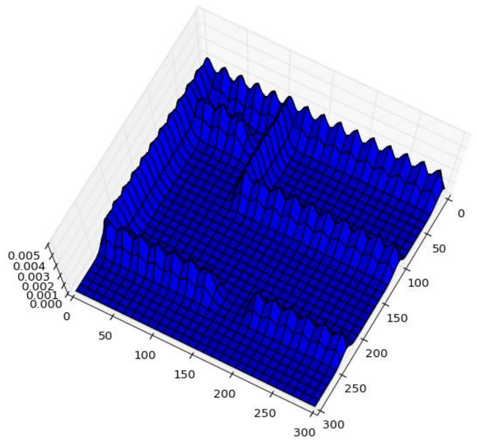


Figure 14: Example VIPARS Map Representation.

A map is defined as an indexed bivariate mixture of Gaussians where $I=[0\ldots X]\times[0..Y]$ and where each member is a Gaussian kernel with covariance $\Sigma[x, y] = \sigma_m^2 I$, and where σ_m represents the map resolution. This corresponds somewhat intuitively with an occupancy grid representation [32], where $w[x,y]$ is related to probability of occupancy for the location (x,y).

During verification, the location random variable (the connection *cp* in eq. (16)) represents the location of the robot for *all* possible executions. It's relevant to compare this with the representation of robot location in a localization algorithm: the representation there may also be a random variable, but the interpretation is different. In any single execution, the robot can really only be at a *single* physical location; the localization distribution is an estimate of this. In verification, the objective is not to find the single most likely location, but to propagate the effects of being at all locations. Rather than using a ray trace algorithm to determine how each location is supported by sensor readings and refining the position estimate based on that, the ray trace algorithm is used by the **MB_Laser** process to gather *all* possible sensor readings that can arise due to the robot location distribution.

5.4 *Modeling Localization*

The first approach involves modeling localization at a high level: modeling not the actual collection of sensory data that produces improved position estimates, but just position estimates that improve with time according to some parameterization. This has the advantage that different localization algorithms can be 'modeled' in verification by just changing the parameterization, not requiring as many hours of expert effort as implementing a new localization algorithm directly in the formal framework. It has the disadvantage that it decouples the localization from predicted sensor measurements, and may miss the effect of measurements that greatly improve or degrade the localization estimate.

The second approach involves the incorporation of existing localization code *directly* into the VIPARS verification algorithm. Localization code, like any program, when executed, will yield once possible trace of a robot mission, whereas VIPARS needs to use that code to probabilistically reason about all executions that are possible given the a-priori environment model information. Our approach considers the embedded code to be capable of transforming a sample from a PARS random variable, and we define a framework for sampling and reconstructing variable distributions. This approach has the advantage of verifying the actual preexisting, localization code that will get executed by the robot at run-time for the mission. It has the disadvantage of potentially lengthening verification times, since multiple samples need to be evaluated for a representative result.

5.4.1 *High-level Model Approach*

Localization starts with the odometry estimate of position at time step t, $q(t) \sim MG$. Through comparisons of sensory returns and the map, it refines the odometry estimate, bringing it closer to the actual position of the robot at time t, $p(t) \sim MG$. At any time, therefore the localization position is some combination of the odometry and the actual position:

$$\ell(t) = (1-k(t))\, p(t) + k(t)\, q(t) \qquad (17)$$

where $k(t) \in [0,1]$ is a time varying gain with $k(t_0)=1.0$, forcing localization to start with just the odometry estimate. The improvement of localization with time is modeled by a monotonic-decreasing dynamics for k:

$$k(t+\Delta t)= t_c\, k(t) \qquad (18)$$

For time constant, $t_c \in [0,1]$, determined from calibration measurements of the localization algorithm to be verified.

5.4.2 *Sampling Approach*

Consider that a preexisting C++ program we want to include in a mission is **P**. A PARS process wrapper for **P** is built, so the code behaves like a 'black box' process $\mathbf{P}\langle x\rangle\langle y\rangle$. Then, like every PARS process, it has an associated flow function $f_P(x)=(y)$ which is calculated by VIPARS. However, when **P** is called, it will map one input value x to an output, y; only one possible execution of **P**, whereas verification has to check *all* possible executions. So, this approach to embedding **P** doesn't work, but, embedded code can *only* be called in this way.

Our approach is to define an extension to the flow function f_P from the process/program **P**: the mixture extended flow function F_P takes a random variable x as input and produces a random variable **y** as output. It samples the input distribution x and calls f_P on the samples, and reconstructs the output distribution mixture $p(y \mid x)= F_P(\boldsymbol{x})$ from the result.

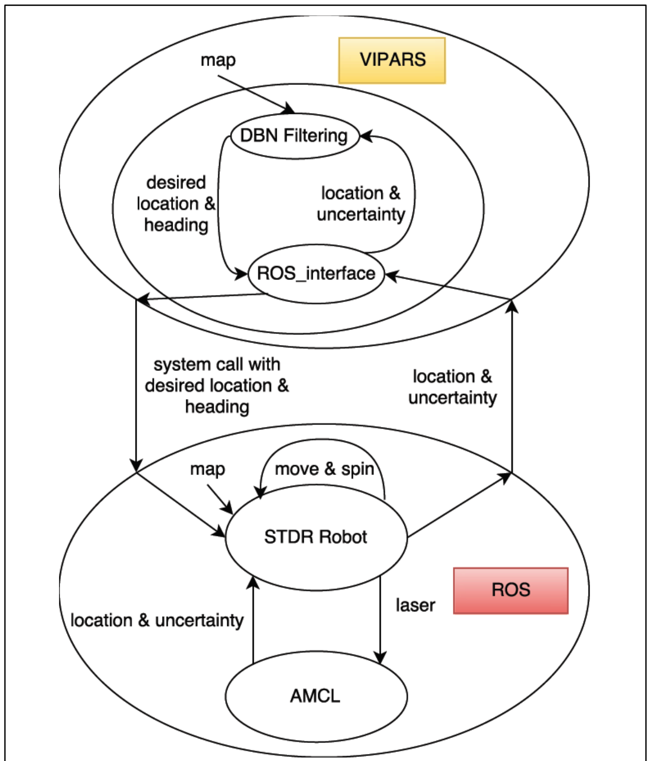


Figure 15: VIPARS-ROS Architecture.

Definition. Let $f_P(x)=y$ be the flow-function for the code to be embedded in verification, defined only by executing that code. Let $\boldsymbol{x}, \boldsymbol{y} \sim MG(CM)$ be random variables over the type of the variables x, y which we denote T. The *mixture extended flow function* (MEF) F_P is defined as follows.

- f_P: $T \rightarrow T$, where $y=f_P(x)$, for $x, y \in T$,
- F_P: MG→MG, where $\mathbf{y} = F_P(\boldsymbol{x})$, for $\boldsymbol{x}, \boldsymbol{y} \in$MG (where MG is the set of all MG), and
- where we define $\mathbf{y}=\mathbf{x}$
- except $\boldsymbol{\mu}(y_i) = f_P(\boldsymbol{\mu}(x_i))$ for all x_i in $\boldsymbol{x}$, and
- where $\boldsymbol{\sigma}(y_i)$ is calculated as follows:
    - $\boldsymbol{\mu}'_j = f_P(s_j)$ for s_i a sample of the input x_i
    - $\boldsymbol{\sigma}(y_i)= \sum_{j=1}^{k} N(s_j; \mu(x_i), \sigma(x_i)) \left((\mu'_j - \mu(y_i))^2 \right)$

The MEF preserves number of members ($|\mathbf{y}|=|\mathbf{x}|$). Each mean is transformed directly $\boldsymbol{\mu}(y_i) = f_P(\boldsymbol{\mu}(x_i))$, requiring multiple executions of the embedded code. Finally, each variance is calculated by carrying out further sample executions for each member $\boldsymbol{\mu}'_j = f_P(s_j)$.

5.5 *Embedding ROS AMCL Localization*

The localization algorithm used in this paper was Adaptive Monte Carlo Sampling (AMCL) [33] as implemented in ROS. In the sampling approach, the DBN filtering engine of VIPARS issued requests to a ROS-based AMCL server to evaluate the MEF function for **Localization**. The interaction is shown in Fig. 15: Whenever the flow function for the

Localization process needed to be evaluated on a position random variable, the position variable was sent from the DBN filtering engine (Top, Fig. 15) via a pipe to a concurrently running ROS system (Bottom, Fig. 15). The STDR simulator node was instructed to move and rotate ("move and spin" in Figure 15) the robot to the appropriate position, and localization data collected from the AMCL node. For simplicity, the MEF function was restricted to single member variables, and rather than calculating the variance by evaluating multiple samples, only the mean value was transformed and the variance calculated by convolving the mean with a zero-mean distribution $N(0, \sigma_s)$. This simplified the hysteresis issue with calling AMCL.

5.6 Verification

Both verification approaches were applied to both waypoint missions. For the high-level approach, **Localization** in eq. (16) implemented eqs. (17), (18) with the gain parameter $t_c = 0.99$. This value was empirically determined from experimentation running ROS AMCL on a *Pioneer 3-AT* robot, carrying out a series of short waypoint missions.

a. Robot moving toward 1st waypoint

b. Robot moving toward 2nd waypoint

d. Robot moving toward goal location

c. Robot after turning a corner

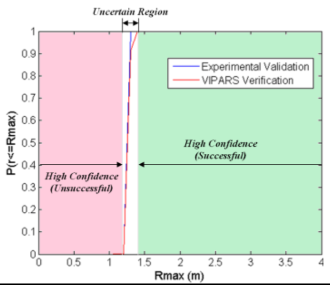
Figure 16: Snapshots of Validation for *Mission-B.*

The sample-based approach implemented the architecture of Figure 15 using ROS version Indigo. A third, odometry-only version of the mission was also run through VIPARS for comparing with both localization methods, and determining whether localization was really necessary for mission success. No additional validation was done on the odometry-only version since that just replicates our prior work [1].

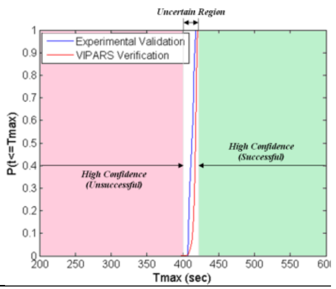
The results of carrying out verification using both approaches with both waypoint missions was a set of performance graphs showing the predicted performance of the missions with respect to the performance criteria.

5.6.1 Validation

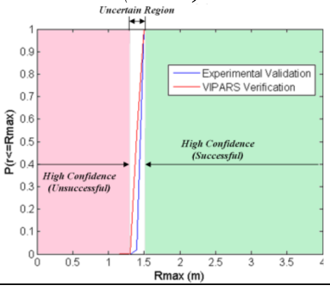
The robot used for the experimental trials is the *Pioneer 3-AT*, a four-wheeled skid-steered mobile robot. The robot is also equipped with a forward-facing SICK laser scanner. The complete validation experiment consists of 50 trial runs for each waypoint mission respectively, which resulted in a total of 100 trial runs. Snapshots of the waypoint mission B are shown in Figure 16. For each trial, mission completion time and relative distance to goal on completion were measured.



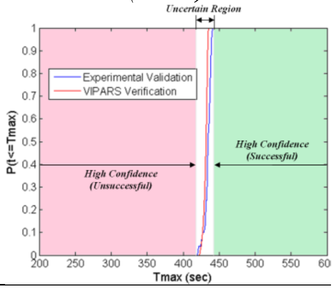
a. *Mission A Spatial Criterion* $P(r \leq R_{max})$



b. *Mission A Time Criterion* $P(t \leq T_{max})$



c. *Mission B Spatial Criterion* $P(r \leq R_{max})$

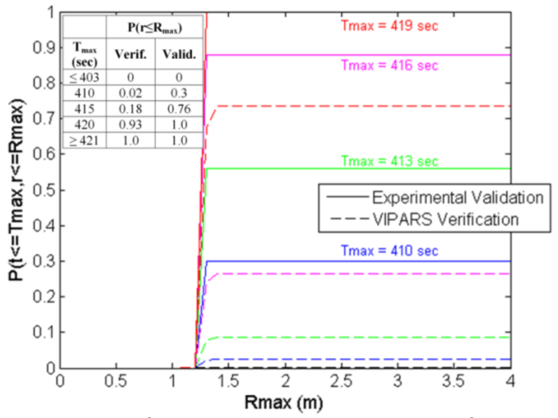


d. *Mission B Time Criterion* $P(t \leq T_{max})$

Figure 17: Results of VIPARS Verification and Experimental Validation of Spatial and Time Performance Criteria for Waypoint Missions A and B. Figures 17a & 17b show the V&V results of spatial and time performance respectively for Mission-A, where the results are divided into three regions based the performance guarantees: *High Confidence (Unsuccessful), Uncertain*, and *High Confidence (Successful)*. Figures 17c and 17d show the V&V results for Mission-B.

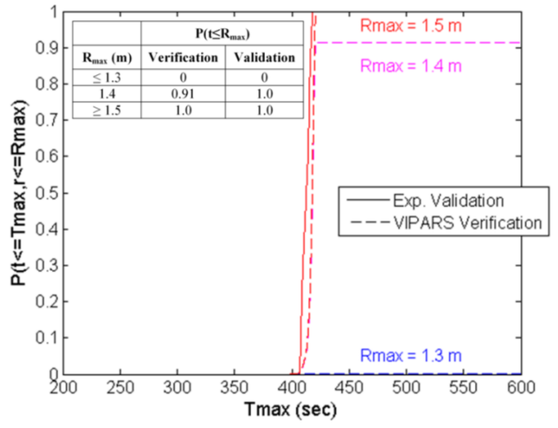
5.6.2 Comparison of verification and validation results

Figure 17 shows the validation results of the performance guarantees for the two waypoint missions. These results are obtained with the sampling-based model of probabilistic localization. Figures 17a and 17c show the V&V results for the spatial criteria $P(r \leq R_{max})$, the probability that the robot arrives within R_{max} radius of its goal location. Figures 17b and 17d show the comparisons for the time criteria $P(t \leq T_{max})$, the probability that the waypoint mission is completed under the time limit, T_{max}. The results illustrate that the VIPARS verification of performance guarantees are consistent with the outcomes from experimental validation. The V&V results can be divided into three regions for further interpretation as before: *High Confidence (Unsuccessful), Uncertain*, and *High Confidence (Successful)* region. Consequently, the mission operator's decision for robot deployment can be based on which region of the mission criteria fall into. For instance, if the specified performance criterion falls within the *Unsuccessful* region (e.g., $R_{max}=0.5m$), the operator can either abort the mission or modify mission parameters and reevaluate. The overall mission success, eq. (15), is defined in terms of both spatial and time criteria. Thus, we examined further in Figures 18 and 19 the effects of various combinations of spatial and time criteria (R_{max} and T_{max}) on the mission success and verification error. The results can also be used to answer queries regarding the performance guarantee for a specific combination of T_{max} and R_{max}. Figure 18 shows the effects of the time criterion T_{max} on the V&V results of the spatial criterion $P(r \leq R_{max})$ for *Mission A*. While the T_{max}'s in both of its high confidence regions (Fig. 17b) have no effect on the verification error for $P(r \leq R_{max})$, T_{max}'s that are in the *Uncertain* region (e.g., $T_{max} = 415$ *sec*) incur significant verification errors. For instance, for $T_{max}=415$ *sec*, VIPARS predicted a success probability of 0.18, while the robot was actually successful 76% of the time in experimental trials. Figure 19 shows the effects of the spatial criterion R_{max} on the V&V results of the time criterion $P(t \leq T_{max})$. While similar observations can be made here as in Figure 18, in this case, R_{max}*'s* have much less impact on the verification error of $P(t \leq T_{max})$ due to VIPARS's accuracy in predicting the spatial performance of mission even in the uncertain region (as shown in Fig. 17a). Nonetheless, our conclusion is that missions with performance criteria in the *Uncertain* regions should generally be avoided.



	$P(r \leq R_{max})$	
T_{max} (sec)	Verif.	Valid.
≤ 403	0	0
410	0.02	0.3
415	0.18	0.76
420	0.93	1.0
≥ 421	1.0	1.0

Figure 18: V&V of Spatial Criterion at various T_{max} for *Mission A*



	$P(t \leq R_{max})$	
R_{max} (m)	Verification	Validation
≤ 1.3	0	0
1.4	0.91	1.0
≥ 1.5	1.0	1.0

Figure. 19: V&V of Time Criterion at various R_{max} for *Mission A*

Lastly, we have also examined the different verification results of VIPARS based on how the probabilistic localization mechanism is modeled: sampling-based and high-level model-based. These results are also compared in Mission A to the verification result for the case when *only odometry information* is used for localization. These verification results are shown in Figures 20-21 along with the validation result for *Mission-A*. While the verification results for different localization modeling approaches are comparable for the time criterion (Fig. 20), the performance based on the sampling-based model is more closely aligned with the validation result for both spatial and time criteria. If only high-probability results are of interest, then the simpler and faster, model-based localization produces acceptable results.

The odometry-only Mission B was 100% unsuccessful during verification due to collisions. However, with the final waypoints moved just 15 cm, the odometry-only mission finishes successfully. Because a small modification enables the odometry-only mission to be potentially successful, it is also clear that localization is not always required for mission success. A contribution of our approach is that it is now possible to answer whether localization is of mission benefit using the performance graphs below in conjunction with the specific performance values of R_{max} and T_{max}. Being able to omit software modules (such as localization) can yield lighter and faster mission code.

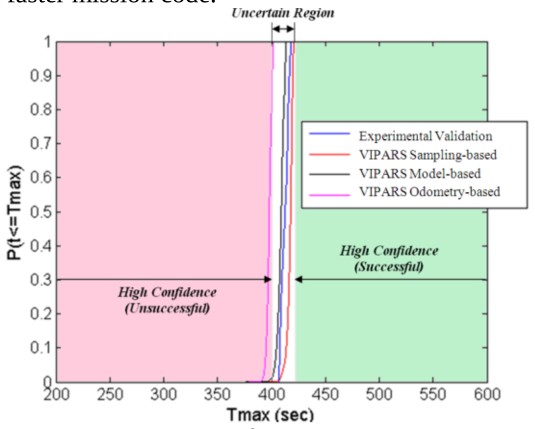


Figure 20: V&V of Time Criterion and Models of Localization

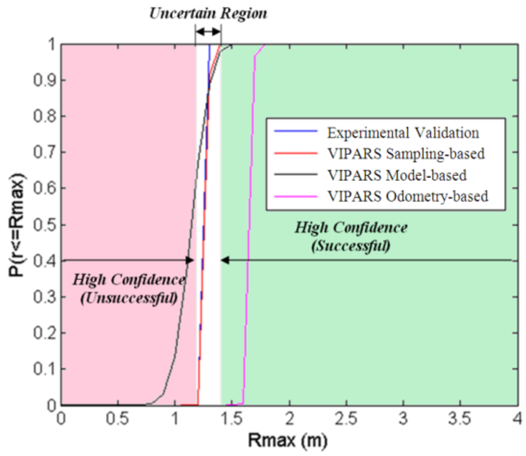


Figure 21: V&V of Spatial Criterion and Models of Localization

6 Conclusions

If teams of autonomous robots are to undertake critical missions such as C-WMD missions, then it is vital to be able to establish performance guarantees for them. This paper addresses the challenge of verifying mission software for autonomous robots that will operate in partially known environments. The approach taken in this paper, and its predecessor [1], differs from common approaches to robotic software verification in two important ways: it emphasizes the roles of a separate but communicating environment model, and it eschews an explicit exploration of the state space of the system combined mission software and environment model for reasons of avoiding state-space explosion. This paper significantly expands [1], which addressed uncertainty in robot motion and sensing, by addressing uncertain geometry in the environment model.

Two classes of mission were investigated: a mission where a team of two robots executes a coordinated set of motions during which they may encounter obstacles whose existence and location is uncertainly known in advance; and a robot mission in which the robot navigates a series of waypoints leveraging probabilistic localization. Approaches to representing and analyzing both mission classes were presented. In addition, separately collected experimental validation results were presented for both classes of mission and were compared to the results from verification.

The comparison of experimental validation and the output of the verification software show the effectiveness of the verification framework in providing performance guarantees for multi-robot missions operating in an uncertain environment. Some of the noted discrepancies between verification and validation may be due to calibration inaccuracies but also the precision limitation from pruning CMG variables.

The colored mixtures developed here may have wider applications. Algorithms that selectively modify mixture members (e.g., image background update [34], in addition to those discussed here) can thus easily propagate subpopulations of one or more members identified for later processing. With respect to complexity and scaling: The computation of *s(t)~CMG* just increases linearly with each additional obstacle (and robot), but each robot must evaluate its own copy. The number of members increase exponentially with each filtering step. In this paper, they were pruned on weight to a maximum number (here 10).

The multiple robot synchronization in this paper involved direct and indirect (i.e., through the environment model) interactions. The robots directly exchanged synchronization messages as they completed each mission bound. Although no communication latency or error was modelled here, it is a straightforward extension to model for example, WiFi limitations. The indirect interaction was limited to the robots being able to view each other as obstacles. While this generalizes easily to any number of robots (the principal complexity is just evaluating eq. (12)) it does not model physical interaction between the robots such as one pushing the other or both physically collaborating in a task.

The paper also addressed verification of missions with probabilistic localization. Two approaches to modeling localization were presented and evaluated: a high-level approach in which only position estimate improvement is modeled, and a sample-based approach, in which the run-time localization code is embedded in verification. Extensive experimental validation is reported for two different waypoint missions using localization. The sample-based approach yields the more accurate estimate, even for the sampling simplification made in this paper. While there is support for the intuition that localization is an asset to mission performance (100% failure of the non-odometry mission; Mission B of Section V.F); a minor modification of 15cm will allow the mission to be verified successful, indicating that the need for localization is mission-specific.

A verification tool is only as effective as its usability [35]. Therefore, a key future direction for this work is the challenge of presenting verification results to the mission designer in an intuitive and effective way. A second thrust of continuing work is the extension and evaluation of this approach for missions that include a human in the loop element.

REFERENCES

[1] D. Lyons, R. Arkin, S. Jiang, T.-L. Liu and P. Nirmal, "Performance Verification for Behavior-based Robot Missions," *IEEE Trans. on Robotics,* vol. 31, no. 3, 2015.

[2] D. MacKenzie, R. Arkin and R. Cameron, "Multiagent Mission Specification and Execution," *Autonomous Robots,* vol. 4, no. 1, pp. 29-52, 1997.

[3] R. Jhala and R. Majumdar, "Software Model Checking," *ACM Computing Surveys,* vol. 41, no. 4, 2009.

[4] P. Trojanek and K. Eder, "Verification and testing of mobile robot navigation algorithms," in *IEEE/RSJ Int. Conf on Intelligent Robots and Systems (IROS*, Chicago, 2014.

[5] D. Walter, H. Taubig and C. Luth, "Experiences in Applying Formal Verification in Robotics," in *29th International Conference on Computer Safety, Reliability and Security*, Vienna Austria, 2010.

[6] L. Kiekbusch, C. Armbrust and K. Berns, "Formal Verification of Behavior Networks including Sensor Failures," *Robotics and Autonomous Systems,* vol. 74, pp. 331-339, 2015.

[7] S. Jiang and R. Arkin, "SLAM-Based Spatial Memory for Behavior-Based Robots," in *11th IFAC Symposium on Robot Control (SYROCO)*, Salvador, Brazil, 2015.

[8] L. DeMoura and N. Bjorner, "Satisfiability Modulo Theories: Introduction and applications," *CACM,* vol. 54, no. 9, pp. 54-67, 2012.

[9] A. Cowley and C. Taylor, "Towards Language-Based Verification of Robot Behaviors," *IEEE/RSJ Int. Conference on Int. Robots and Systems (IROS)*, 2011.

[10] Ropertz, T. and R. Berns., "Verification of behavior-based networks-using satisfiability modulo theories," in *ISR/Robotik 2014; 41st Int. Symposium on Robotics*, 2014.

[11] M. Kim, K.-C. Kang and H. Lee, "Formal Verification of Robot Movements - a Case Study on Home Service Robot SHR100," in *IEEE Int. Conf. Rob. & Aut.*, 2005.

[12] M. Webster, C. Dixon, M. Fischer, M. Salem, J. Saunders, K.-L. Koay and K. Dautenhahn, "Formal Verification of an Autonomous Personal Robotic Assistant," in *AAAI 2014 Symposioum Modeling in Human-machine Systems: Challenges for Formal verification*, Stanford CA, 2014.

[13] M. Fisher, L. Dennis and M. Webster, "Verifying Autonomous Systems," *CACM,* vol. 56, no. 9, pp. 84-93, 2013.

[14] M. Guo, K. Johansson and D. Dimarogonas, "Revising Motion Planning Under temporal Logic Specifications in Partially Known Workspaces," *IEEE. int. Conf. Rob. & Aut.*, 2013.

[15] S. Sarid, B. Xu and H. Kress-Gazit, "Guaranteeing High Level Behaviors while Exploring Partially Known Maps," in *Rob. Sc. & Sys.*, 2013, 2008.

[16] M. Proetzsch, K. Berns, T. Schuele and K. Schneider, "FORMAL VERIFICATION OF SAFETY BEHAVIOURS OF THE OUTDOOR ROBOT RAVON," in *4th Int. Conf. on Informatics, Aut. and Control* ., Dortmund, Germany, 2007.

[17] A. Zaks and R. Joshi, "Verifying Multi-threaded C programs with SPIN," in *15th International SPIN Workshop*, Los Angeles CA, 2008.

[18] M. O'Brien, R. Arkin, D. Harrington, D. Lyons and S. Jiang, "Automatic Verification of Autonomous Robot Missions," in *4th Int. Conf. on Simulation, Modelling and Prog. for Aut. Robots*, Bergamo, Italy, 2014.

[19] H. Kress-Gazit and G. Pappas, "Automatic Synthesis of Robot Controllers for Tasks with Locative Prepositions.," in *IEEE Int. Conf. on Rob. & Aut.*, Anchorage, Alaska, 2010.

[20] S. Bensalem, K. Havelund and A. Orlandini, "Verification and validation meet planning and scheduling.," *Int. Journal on Software Tools for Technology Transfer.,* vol. 16, no. 1, pp. 1-12, 2014.

[21] D. MacKenzie and R. Arkin, "Evaluating the Usability of Robot Programming Toolsets," *Int. Journal of Robotics Research,* vol. 4, no. 7, pp. 381-401, 1998.

[22] R. C. Arkin, Behavior-Based Robots, Cambridge MA: MIT Press, 1998.

[23] A. Roscoe, The theory and practice of concurrency, Prentice-Hall, 1997.

[24] T. Bolognesi and E. Brinksma, "Introduction to the ISO Specification Language LOTOS," *Computer Networks & ISDN Sys,* vol. 14, no. 1, pp. 25-59, 1987.

[25] M. Steenstrup, M. Arbib and E. Manes, "Port Automata and the Algebra of Concurrent Processes," *JCSS,* vol. 27, no. 1, pp. 29-50, 1983.

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

[27] S. Russel and P. Norvig, Artificial Intelligence, Prentice-Hall, 2010.

[28] D. Lyons, R. Arkin, S. Jiang, D. Harrington, F. Tang and P. Tang, "Probabilistic Verification of Multi-Robot Missions in Uncertain Environments," in *IEEE Int. Conf. on Tools with AI*, Vietro sul Mare, Italy, 2015.

[29] R. Szczerba and B. Collier, "Bounding overwatch operations for robotic and semi-robotic ground vehicles," in *SPIE Aerospace Conference on Guidance and Navigation*, 1998.

[30] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bulletin of the Calcutta Mathematical Society,* no. 35, pp. 99-109, 1943.

[31] F. Dellaert, D. Fox, W. Burgard and S. Thrun, " Monte Carlo localization for mobile robots," in *IEEE Int. Conf. on Rob. & Aut.*, Detroit, 1999.

[32] A. Elfes, "Using Occupancy Grids for Mobile Robot Perception and Navigation," *Computer,* pp. 48-57, April 1989.

[33] D. Fox, "KLD–Sampling: Adaptive Particle Filters," in *Neural Information Processing Systems 14 (NIPS)*, Vancouver Canada, 2001.

[34] C. Stauffer and W. Grimson, " Adaptive background mixture models," in *CVPR*, 1999.

[35] M. O'Brien and R. Arkin, "An Analysis of Displays for Probabilistic Robotic Mission Verification Results," in *7th International Conference on Applied Human Factors and Ergonomics*, Las Vegas NV, 2016.