# Probabilistic Verification of Multi-Robot Missions in Uncertain Environments

Damian M. Lyons*, Ronald C. Arkin‡, Shu Jiang‡, Dagan Harrington*, Feng Tang*, Peng Tang*

*Fordham University
New York USA
dlyons@fordham.edu

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

*Abstract*— **The effective use of autonomous robot teams in highly-critical missions depends on being able to establish performance guarantees. However, establishing a guarantee for the behavior of an autonomous robot operating in an uncertain environment with obstacles is a challenging problem. This paper addresses the challenges involved in building a software tool for verifying the behavior of a multi-robot waypoint mission that includes uncertain environment geometry as well as uncertainty in robot motion. One contribution of this paper is an approach to the problem of a-priori specification of uncertain environments for robot program verification. A second contribution is a novel method to extend the Bayesian Network formulation to reason about random variables with different subpopulations, introduced to address the challenge of representing the effects of multiple sensory histories when verifying a robot mission. The third contribution is experimental validation results presented to show the effectiveness of this approach on a two-robot, bounding overwatch mission.**

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

## I. INTRODUCTION

It is crucial that a team of autonomous robots on a mission to search for and identify weapons of mass destruction (WMD) respond appropriately to their environment and each other to accomplish their task given the high cost of mission failure in that scenario. This paper presents an approach to building a software tool for a-priori automatic verification of software for a multirobot team operating in an uncertain environment.

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 [1]. It differs in that a robot program continually interacts with its uncertain and dynamic environment, which has to be included in the verification problem. However, following general purpose software verification [1], many robot software verification papers do not include any model of the environment in which the mission is carried out and verify properties such as absence of deadlock or run-time errors [2] [3]. 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. A model of the environment is necessary for these.

In some cases, the properties to be verified are used themselves to implicitly express the designer's knowledge (or expectation) of environment dynamics [4]. This seems like an informal and somewhat error-prone way to capture environment dynamics. Some of the most recent work does include environment models: The UK EPSRC-funded project on Trustworthy Robotic Assistants recently proposed representing unstructured environment using the Brahms [5] 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 has been identified as one of the key 'lessons learned' in applying standard formal techniques to robot missions [3]. Related work 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 [6], 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" [7] – 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.

In prior work, we have developed an approach to the verification of behavior-based multirobot missions that include robot motion uncertainty [8] [9]. In that approach, the mission designer builds the robot program using the graphical *MissionLab* [10] [11] mission design editor. The mission is automatically translated [12] into an internal process algebra (PARS – Process Algebra for Robot Schemas) from which static analysis algorithms [13] extract a set of probabilistic relations for the program and environment variables. A Bayesian Network approach is constructed from these relations and used to verify the performance of the program. Experimental validation has shown that verification results for this approach correspond closely to real mission results. However, while that work captured uncertainty in robot motion, it did not account for other environment interactions such as with obstacles.

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 to reason about random variables with different subpopulations.

The next section describes the *MissionLab* tool and the process of mission performance verification. Section III addresses the challenge of multiple robots and uncertain obstacle avoidance, presenting our proposed approach based on tagging subpopulations of Gaussian mixture models. Section IV presents the verification and experimental validation of a multirobot mission in an area strewn with obstacles whose locations are uncertainly known a-priori.

## II. MISSIONLAB WITH VERIFICATION

This section reviews building robot software with *MissionLab*[1] [10] and introduces the *Bounding Overwatch* multirobot mission which will be our running example.

### A. Mission Design

A mission designer can use *MissionLab* to design robot behavior with its usability-tested [10] graphical programming frontend, as shown in Figure 1.

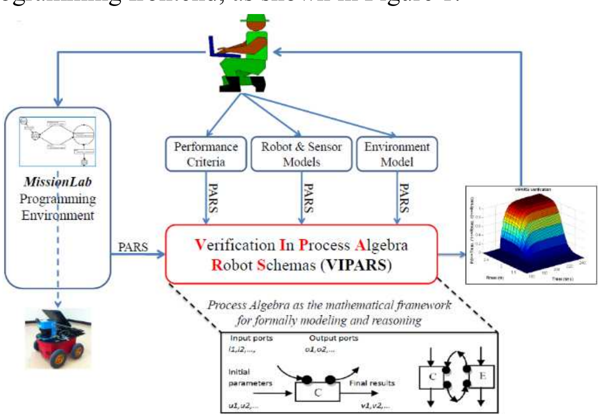



**Figure 1.** *MissionLab*/VIPARS System Architecture.

The VIPARS (*Verification in PARS*) [13] module provides a performance verification functionality for *MissionLab*. The inputs for VIPARS are the mission program as designed in *MissionLab*'s *CfgEdit* graphical interface, a set of designer selected library models of the robot, the sensor systems, the mission operating environment, and the mission performance criteria. For example, an operator might design a single-robot, waypoint mission to take place in a moderately-cluttered warehouse and to 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) and verify that the mission is a success, given the above, for some threshold probability.

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

The *MissionLab* mission is automatically translated to PARS [12], the formal language of VIPARS. The designer then selects which library models of Pioneer 3-AT, sonar and gyroscope, and moderately cluttered indoor environment to use. These library 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.

VIPARS predicts whether the mission will achieve the performance criteria 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.

### B. Multirobot Mission with Uncertain Obstacles

*Bounding overwatch* is a military movement tactic used by units of infantry to advance forward under enemy fire or when crossing dangerous areas where enemy encounter is expected [14]. This strategy can be used by robots to move stealthily inside a building to search for biohazards which may be guarded by hostile forces.

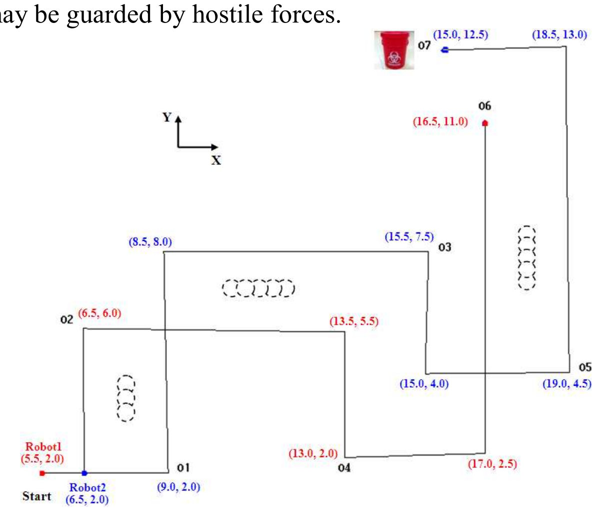



**Figure 2.** Bounding Overwatch with Two Robots

In the *bounding overwatch* mission depicted in Figure 2, two robots coordinate their movements in a "leapfrogging" manner while advancing toward a biohazard. The mission proceeds with Robot2 bounding toward ***O1***, the first ***O****verwatch* position. Once Robot2 reaches ***O1***, it sends a *"Cleared"* message to Robot2 indicating that it is safe for Robot1 to proceed. Robot1 then bounds to ***O2*** and sends the *"Cleared"* message to Robot2; then Robot2 bounds to ***O3***, and so on. The mission ends with Robot2 at ***O7***, near the biohazard. In prior work [8], we successfully verified and experimentally validated a version of this mission with an environment model that included robot motion uncertainty. However, the environment model had no obstacles.

In this paper, the operating environment of the mission includes some obstacles whose exact locations, and even their existence, are not known with certainty in advance. All that is known in advance is that the obstacles may (or may not) be somewhere within the locations (illustrated with dashed circles) shown in Figure 2. 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 the robots.

The behaviors of Robot1 and Robot2 are specified in *MissionLab* as behavioral finite state automata (FSAs). Each behavioral FSA consists of *GoToGuarded, NotifiedRobots, Spin*, and *Stop* behaviors and *AtGoal, HasTurned, Notified, and MessageSent* triggers for state transitions. The behavioral FSA of Robot1 is shown in Figure 3; the FSA for Robot2 (omitted for brevity) is similar.

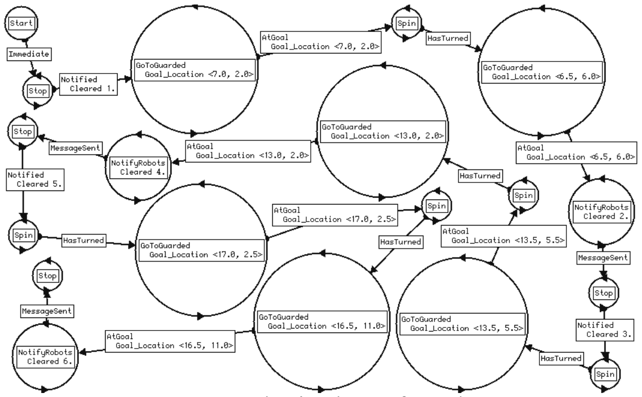



**Figure 3.** Behavioral FSA for Robot1

This automaton represents high level robot behaviors, hiding important physical details such as obstacle avoidance and its relation to goal-oriented motion. The behavioral FSA is translated to a MissionLab internal language called CNL [11] which does contain all this information. A translator from CNL to the process algebra PARS [12] produces a model of the program which does have all the detail for verification. For the *bounding overwatch* mission, the following performance criteria are used to evaluate mission performance:

1. $\mathbf{R_{max}}$ – the success radius; each robot is required to be within this radius (e.g., 1.0 *m*) of its goal location in the physical environment
2. $\mathbf{T_{max}}$ – the maximum allowable time; the mission is required to be completed under this time limit (e.g., 150 *s*)

Overall mission success is defined as:

$$\textbf{Success} = (\mathbf{r_1 \le R_{max}}) \textbf{ AND } (\mathbf{r_2 \le R_{max}}) \textbf{ AND } (\mathbf{t \le T_{max}}) \qquad (1)$$

Where $r_1$ and $r_2$ are Robot1's and Robot2's relative distances to their respective goal locations (i.e., ***O6*** and ***O7*** in Figure 2), and ***t*** is the mission completion time. That is, the *bounding overwatch* mission is only considered successful when both robots are within $\mathbf{R_{max}}$ radius of their respective goal locations and when they complete the mission under $\mathbf{T_{max}}$ seconds.

## III. Verification with Uncertain Geometry

The PARS process algebra, extraction of flow functions and Bayesian Network filtering has been described in prior papers, and the reader is referred there for details [13] [9]. The bounding overwatch multirobot mission, without obstacles, is described in more detail in [8]. The following subsection reviews enough of this background to describe our results.

### *A. PARS*

PARS is a process-algebra for representing and analyzing robot programs interacting with their environment. A process $\boldsymbol{P}$ is written as:

$$\boldsymbol{P}\langle \boldsymbol{u_1}, \dots, \boldsymbol{u_n}\rangle(\boldsymbol{i_1}, \dots, \boldsymbol{i_j})(\boldsymbol{o_1}, \dots, \boldsymbol{o_k})\langle \boldsymbol{v_1}, \dots, \boldsymbol{v_m}\rangle \qquad (2)$$

where $u_1,\dots,u_n$ are the 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 $v_1,\dots,v_m$ are final result values generated by the process. Processes can compute results from initial values, but these results may also be influenced by any communications that occur over the port connections during computation. Port connections can be used to represent the points of interaction between a controller and its environment as well as the usual message passing between components of a controller (or environment model). Process variables can be of a variety of data types and can be random variables.

Processes are either *atomic* or *composite.* Composite processes are algebraic combinations of other processes using composition operators: parallel ('|'), disabling ('#') and sequential (';'). Unlike many process algebras, there is no 'choice' operator in PARS. A sequential chain of processes, e.g., $\mathbf{Eq}\langle x,y\rangle$ ; $\mathbf{P}$, terminates for the first process that has a termination status of abort (e.g. in this case, if $x \neq y$, $\mathbf{P}$ is not reached because the condition process $\mathbf{Eq}$ aborts). Bounded recursion is captured using tail-recursive (TR) process definitions, written for example:

$$\mathbf{P}\langle x\rangle = \mathbf{Q}\langle x\rangle\langle y\rangle \,;\, \mathbf{P}\langle y\rangle \qquad (3)$$

Eq. (3) defines a process $\mathbf{P}$ that repeats process $\mathbf{Q}$ (until $\mathbf{Q}$ aborts, at which point $\mathbf{P}$ terminates, returning its results). A variable *flow function* ($f_P$) is associated with each $\mathbf{P}$ that maps the values of the variables of $\mathbf{P}$ at the start of each recursive step 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 $\mathbf{P}$ does not abort.

The system to be verified is expressed in PARS as the parallel, communicating composition (**Sys)** of robot controller processes (**Ctr**) and environment model processes, (**Env**) shown as an example here:

$$\begin{aligned} \mathbf{Sys}\langle r_1,r_2\rangle &= \mathbf{Ctr}\langle r_1\rangle(a)(b) \mid \mathbf{Env}\langle r_2\rangle(b)(a) \qquad (4)\\ &= \mathbf{Sys'}\langle r_1,r_2\rangle \,;\, \mathbf{Sys}\langle f_{Sys}(r_1,r_2)\rangle \end{aligned}$$

$$f_{Sys}(r_1,r_2) = (f_{Sys,r1}(r_1,r_2),\ f_{Sys,r2}(r_1,r_2)) \qquad (5)$$

The input of **Ctr** is connected to the output of **Env**, ($a$) in eq. (4), while the output of **Env** is connected to the input of **Ctr**, ($b$) in eq. (4). Lyons et al. [13] develops an interleaving theorem and associated algorithm *Sysgen* with linear computational complexity, by which the parallel, connected network of process on the top line of eq. (4) can be converted to the TR process on the second line of eq. (4), and in turn from which *a system flow function*, e.g. (5), can be extracted. When $r_1$ and $r_2$ are random variables, the expressions in (5) describe conditional probability relations (6), relating random values at time $t$ to those at $t+1$. These relations are the basis of a Dynamic Bayesian Network (DBN) [15], a Bayesian network that is used to carry out filtering, forward propagation of probability distributions.

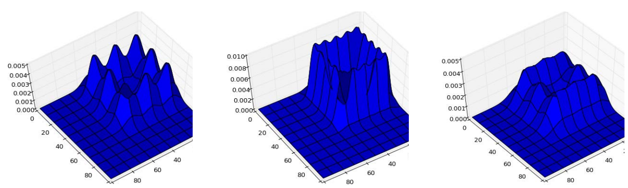
$$f_{Sys,r1}(r_{1,t}, r_{2,t}) = P(r_{1,t+1} | r_{1,t}, r_{2,t}) \tag{6}$$

Random variables are represented as multivariate mixtures of Gaussians, and operations on random variables are automatically translated by VIPARS into operations on distributions [9].

### B. Uncertain Geometry Model

The geometry of the environment in which the robot program will be executed is not completely known in advance. Our approach is to build a probabilistic model of the environment based on any a-priori information. One way to generate such a model is as shown in Figure 2: Several spatial locations along the mission are annotated a-priori with possible obstacles. Another approach would be to use the map output from probabilistic mapping software that has been used to measure the environment, including any dynamic obstacles. However, no matter how the model originates, this approach is based on being able to say something, tagged with probability, about what geometry the environment has.

For the bounding overwatch example, the environment is modelled as collection of isotropic bivariate Gaussian mixtures (Figure 4). Figure 4(a) shows a mixture of 8 members modelling a rectangular 2D obstacle. Figure 4(b) shows the model with 16 members. Anisotropic members can also model asymmetric spatial uncertainty (as required in Figure 2), e.g. Figure 4(c).



(a) (b) (c)

**Figure 4**. Modelling geometry with bivariate Gaussian mixtures.

The *GoToGuarded* behavior states translate to the process network shown in (7).

**Coop** ⟨1,1,1⟩ (vg, vo, vn)(v) |
**Move_to** ⟨P0,G3⟩ (pR)(vg) |
**Noise** ⟨ns⟩ (pR)(vn) |
**Avoid_Obstacles** ⟨r⟩ (pR,obR)(vo) (7)

The **Avoid_Obstacles** process receives the robot position (pR) and sensed obstacles (obR) and generates a potential field based avoidance velocity (vo) [16]. In verification, the position and sensed obstacles variables are (random variable represented by) distributions, and they are calculated by an environment model process based on uncertain geometry modelled as in Figure 4. **Move_to** generates a velocity towards the goal G3 (vg) and **Noise** generates a small velocity perturbation to escape potential minima (vn). The Coop process combines all three into a single command velocity (v) with equal weights (1,1,1). All of these processes are defined in the PARS-CNL library as TR processes and it is possible to automatically extract flow functions for verification.

In execution, the input and output of these processes correspond to the connections of *GoToGuarded* with the real robot and sensors. In verification, this information is provided instead by the network shown in (8).

**Robot**⟨P0, $\boldsymbol{\Delta t}, \boldsymbol{\varphi}$ ⟩ (v)(pR) | (8)
**Sensors**⟨S0,sr,sn⟩ (pR,pE)(s) |
**Geometry**⟨E⟩ (pR,pR2)(pE)

The **Robot** process takes a velocity command and generates a new position distribution according to (9) 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, and $\varphi(t)$ is the position uncertainty – the sum of translational, skittering (translation due to rotation) and rotational uncertainty values, all modeled as bivariate Gaussian distributions and estimated by calibration measurements. The **Robot** process model is described in prior work [13]

$$\boldsymbol{p(t+\Delta t) = p(t) + v(t)\Delta t + \varphi(t)} \tag{9}$$

The **Sensors** process calculates what obstacle locations will be sensed by the robot, implemented as follows:

**Sensors**⟨S0,sr,sn⟩ (pR,pE)(obR) = (10)
**In**⟨pR⟩⟨p⟩ ; **In**⟨pE⟩⟨e⟩ ;
( **Gtr**⟨ d(p,e), sr⟩⟨p1⟩ ; **Out**⟨obR,p1⟩ |
**Lte**⟨ d(p,e), sr⟩⟨p2⟩ ; **Out**⟨obR, sn+p2 ⟩ ) ;
**Sensors**⟨S0,sr,sn⟩ .

The robot position (p) and geometry (e) are input from whatever **Sensors** has been connected to; in this case, the **Robot** process and the **Geometry** process. The latter continually adds the latest position distributions for both robots to the static geometry (obstacles) and transmits this. 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 [17] between robot position and the geometry distribution. This coefficient measures the amount of overlap between two multivariate normal distributions $p$ and $q$ as follows:

$$\boldsymbol{BC(N(\mu_0,\Sigma_0), N(\mu_1,\Sigma_1))} \tag{11}$$

$$\boldsymbol{= \exp\left(-\frac{1}{8}(\mu_0-\mu_1)^T\Sigma^{-1}(\mu_0-\mu_1)\right)\sqrt{\frac{\sqrt{|\Sigma_0||\Sigma_1|}}{|\Sigma|}}}$$

$$\boldsymbol{where\ \Sigma = \frac{|\Sigma_0||\Sigma_1|}{2}}$$

The result of d(.) is a bivariate distribution whose members correspond 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)).

### C. Conlicting Hypothesis Histories

The flow-functions extracted by VIPARS from (7) (and the remainder of the Mission program) and (8) provide the probabilistic relations between variables used to construct a

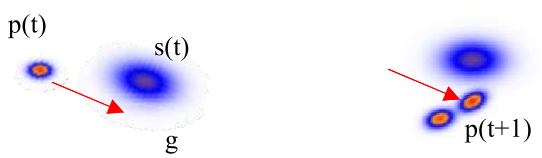
Dynamic Bayesian Network. The verification of the program consists of applying filtering while monitoring for completion of the performance criterion (2) before the mission deadline ($\mathbf{T_{max}}$) is reached. It's necessary to delve into a little more detail about this verification, since we will encounter a problem with the approach so far.

The flow functions automatically extracted by VIPARS from the *GoToGuarded* network (7) 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 heavyside step functions h(.) and unit vector u(.), and translate operations on variables (e.g., addition) to equivalent operations on distributions (e.g., convolution).

$$
\begin{aligned}
f_{vo}(s,p) &= r - h(r - s(t))s(t), \\
f_{vg}(p,g) &= u(\,p(t) - g\,)s_{max} \\
f_v(v_o, v_g\, v_n) &= v_o * v_g * v_n
\end{aligned}
\tag{12}
$$

The obstacle velocity (vo in (7)) is specified by $f_{vo}$ as linearly proportional to the distance to the obstacle $r - s(t)$ 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. Note that the flow functions in (12) just come from whatever program the user has constructed in *MissionLab.*

Consider the example shown in Figure 5: At some time t, the position (p(t), a single member distribution) is close enough to the sensed obstacle s(t) that an obstacle repulsive velocity (vo) is generated in addition to the velocity towards the goal (vg) (Figure 5(a)). The portion of the position distribution that resulted in no obstacle detection (p1 in (10)) should be convolved with just a forward velocity; the portion that had obstacle detection (p2) should be convolved with both forward and repulsion (Figure 5(b)).



(a) (b)

**Figure 5.** Example of Obstacle Avoidance

In fact, there is *insufficient information* in the random variable model as is to correctly represent the situation. During filtering, 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)* MoG 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 MoG member, which originates from *s(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 *s(t)* due to (10).
2) The conditional nature of $f_{vo}$ (i.e., the step-function) means that not every member of *s(t)* generates a repulsive velocity (e.g., because it's too far away).
3) The final, convolution for $f_v$ in (12) will apply goal and repulsion velocities to *all* position modes, not just the ones as show in Figure 5.

### *D. 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 propagate to the sensing distribution, so that it becomes clear how the sensing relates to position. The mixture representation for random variable values 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 $c(a_i)=c_i$. A CMG is evaluated at a point $x$ in the usual way as $CMG(x\,;\,CM)$:

$$
\boldsymbol{CMG}(\boldsymbol{x}; \{(\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i}, \boldsymbol{w_i}, \boldsymbol{c_i}) \mid \boldsymbol{i} \in \mathbf{1}, \ldots, \boldsymbol{m}\}) = \sum_{i=1}^{m} \boldsymbol{w_i}\, \boldsymbol{N}(\boldsymbol{x}; \boldsymbol{\mu_i}, \boldsymbol{\Sigma_i}), \quad \sum_{i=1}^{m} \boldsymbol{w_i} = \mathbf{1} \tag{13}
$$

The color tags now 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 (12) 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 *s(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 $f_v$ (12) 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 $p_2$. This solution also addresses the second complication, since if the step function in $f_{vo}$ trims members from *s(t)*, the members of $v_o(t)$ and $v_g(t)$ 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 validation evidence for this statement.

## IV. VERIFICATION AND VALIDATION

The Overwatch mission presented in Section 3 is verified using the modified CMG filtering and the verification results experimentally validated in this section.

### A. *Mission Verification*

The Overwatch mission was manually translated from the *MissionLab* (m) L language produced by the *MissionLab CfgEdit* (Fig. 3) using the same template approach as the automated translator in [12] (which is currently restricted to single-robot missions). The robot, sensor and environment library models were developed using the CMG representation. The CNL PARS library contains the process implementation of CNL behaviors constructed manually based on inspection of the *MissionLab* CNL C++ library and used in previous missions.

VIPARS minimal output is whether the mission will succeed given the robot, sensor and environment models and the performance criterion. In the interest of providing more than just a binary result, VIPARS can also produce 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).

### B. *Mission Validation*

The objective of validation experiments is to validate that VIPARS' predicted performance for the mission is consistent with the actual performance with physical robots in a real environment. 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 2 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 $\mathbf{R_{max}}$ and $\mathbf{T_{max}}$ 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. $\boldsymbol{t}$ – 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.

### C. *Comparison*

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 6 shows the verification and validation spatial criteria for this mission as the probability both robots are within $\mathbf{R_{max}}$ radius of their respective goal locations $\mathbf{P(r_1 \leq R_{max}, r_2 \leq R_{max})}$ versus $\mathbf{R_{max}}$. The graph has three regions based on VIPARS verification: 1) *High Confidence (Unsuccessful), 2) Uncertain*, and 3) *High Confidence (Successful)*. These regions are defined based on the probability of success from the verification module only since we won't have validation results during actual missions.

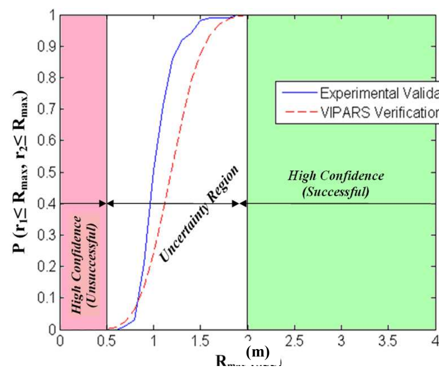



**Figure 6.** Verification vs. Validation of Spatial Criterion $\mathbf{P(r_1 \leq R_{max}, r_2 \leq R_{max})}$

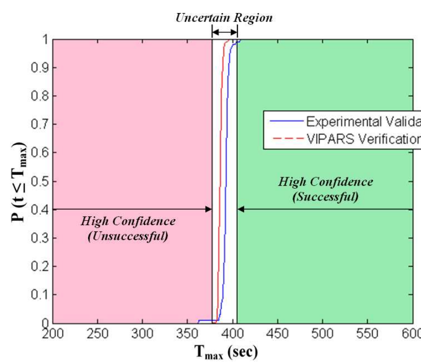
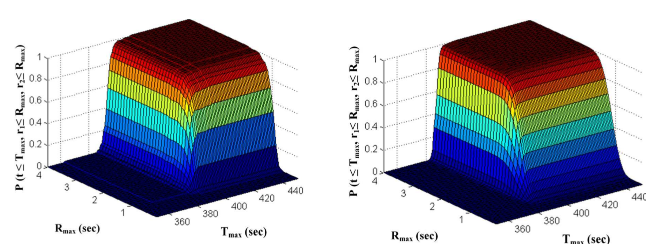



**Figure 7:** Verification vs. Validation of Time Criterion $\mathbf{P(t \leq T_{max})}$




a) Experimental Validation b) VIPARS Verification

**Figure 8:** Validation (a) and Verification (b) of Overall Mission Success $\mathbf{P(r_1 \leq R_{max}, r_2 \leq R_{max}, t \leq T_{max})}$

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. 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., $\mathbf{R_{max}}$) 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. The validation result in Figure 8 further justifies the uncertain nature of this region since the discrepancies between verification and validation are non-zero in the *Uncertain* region.

Figure 7 shows the verification and validation for the time criterion as, the probability that the bounding overwatch mission is completed by **t, P(t≤$T_{max}$)** 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. We also observed some discrepancies outside the *Uncertain* region, near its boundaries. Ideally, we would like all the errors to be within the *Uncertain* region. However, at a closer examination, the errors between the verification and validation success probabilities outside the *Uncertain* region are actually ≤0.01 (i.e., within ~1.01% error). For instance, at the boundary between *Uncertain* and *High Confidence Successful* 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 region since the experimental validation has a success probability of 0.99 and higher.

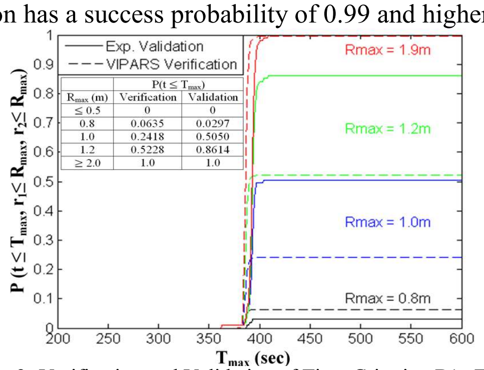



**Figure 9:** Verification and Validation of Time Criterion **P(t≤$T_{max}$)** at various **$R_{max}$**

We have examined individual performance criterion separately thus far. However, the overall mission success (Eq. 1) was defined in terms of both spatial and time criteria. Figure 8 shows the verification and validation of the performance guarantee for the overall mission success, **P($r_1$≤$R_{max}$,$r_2$≤$R_{max}$,t≤$T_{max}$)**, 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 9-10. Figure 9 shows the verification and validation of the time criterion, **P(t≤$T_{max}$)**, at various fixed values of the spatial criterion, **$R_{max}$**. We observed that **$R_{max}$** in both high confidence regions (i.e., **$R_{max}$**≤ 0.5m and **$R_{max}$**≥2.0m, Figure 6) has no effect on **P(t ≤ $T_{max}$)**. However, **$R_{max}$** in the *Uncertain* region (e.g., **$R_{max}$**= 0.8m, 1.0m, 1.2m) have significant impact on **P(t≤$T_{max}$)**. Specifically, **P(t≤$T_{max}$)** plateaus at different probability values for different $R_{max}$'s in the *Uncertain* region. For instance, for **$R_{max}$** of 1.2m, **P(t≤$T_{max}$)** plateaus at 0.5228, which is the value of **P($r_1$≤1.2,$r_2$≤1.2)** for the spatial criterion in Figure 6.

There is a significant discrepancy between verification and validation of **P(t≤$T_{max}$)** when **$R_{max}$**'s are in the *Uncertain* region (max 400 mm). Similar observations are made in Figure 10 for **P($r_1$≤$R_{max}$,$r_2$≤$R_{max}$)** 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 (e.g., use different robots).

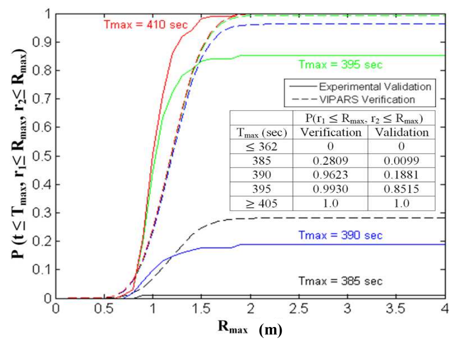



**Figure 10:** Verification and Validation of Spatial Criterion **P($r_1$≤$R_{max}$,$r_2$≤$R_{max}$)** at various **$T_{max}$.**

## V. CONCLUSIONS

This paper has addressed the problem of automatically verifying the behavior of a team of autonomous robot operating in an environment that can include the presence of obstacles, and for which the only a-priori information includes probability distributions of obstacle positions. It builds on prior work [8] [12], where user-designed programs are automatically translated to a set of probabilistic expressions for the evolution of the program variable values, implicitly characterizing the program state-space. Determining whether these expressions can be solved to guarantee a performance criterion is accomplished by applying probabilistic filtering.

A multivariate mixture of Gaussians model is proposed here to model static and dynamic parts of the robot's environment. A modification of the mixture model, the addition of color tags and color-respecting operations, is necessary to ensure that sensory history is preserved during filtering. The proposed approach does allow for the representation of continuous variables and uncertainty in the location and size of objects and obstacles. Furthermore, as a concept, it's not that removed from the probabilistic output of a mapping and localization program. There is little additional related work found by the authors in the verification field to which to compare this. However in the somewhat related field of correct-by-construction, Livingston et al. [18] addresses reactive synthesis for a discretized space, rooms with connecting doors and uncertainty as to their state. Kress-Gazit et al. [19] addresses the problem for discretized environments where paths may be blocked and new paths reactively found.

Of course all these approaches ask that it be possible to say something a-priori about the environment. Verification traditionally considers nondeterminism – any combination of any inputs are possible. Considering any possible combination of obstacles of any size in any location is infeasible. An alternate approach is proposed by Fisher et al. [20], who address the difficulty of specifying a-priori conditions by verifying the robot's belief rather than its actual behavior. But this suffers the same issue as having no environment model: the robot's belief may not correspond to what actually happens (due to environmental dynamics). Guo et al. [21] and Sarid et al. [22] 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.

Color mixtures are a model that may have wider applications. Algorithms that selectively modify mixture members (e.g., image background update [23], 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 $s(t)\sim CMG$ just increases linearly with each additional obstacle (and robot), but each robot has to 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).

Validation results were presented here for the verification of an extended two-robot bounding overwatch mission in an environment with uncertain obstacles. The results 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.

Future work will building on this uncertain geometry model and address the automatic verification of programs that include a probabilistic localization component.

## ACKNOWLEDGMENT

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

## VI. REFERENCES

[1] R. Jhala and R. Majumdar, "Software Model Checking," *ACM Computing Surveys* 41(4) 2009.

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

[3] D. Walter, H. Taubig and C. Luth, "Experiences in Applying Formal Verification in Robotics," in *29th Int. Conf. on Comp. Safety, Reliability and Security* 2010.

[4] 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. and Aut.* 2005.

[5] M. Webster, C. Dixon, M. Fischer, M. Salem, J. Saunders, K.-L. Koay and K. Dautenhahn, "Formal Verification of an Autonomous Personal Robotic Assistant," *AAAI 2014 Symp. Mod. in Human-machine Sys.: Chall. for Formal Verif.* 2014.

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

[7] S. Bensalem, K. Havelund and A. Orlandini, "Verification and validation meet planning and scheduling.," *Int. J. on Soft. Tools for Tech. Trans. 16(1)* pp. 1-12 2014.

[8] D. Lyons, R. Arkin, S. Jiang, D. Harrington and T. Liu, "Verifying and Validating Multirobot Missions," *IEEE/RSJ Int. Conf. on Robots and Systems* 2014.

[9] D. Lyons, R. Arkin, T.-L. Liu, S. Jiang and P. Nirmal, "Verifying Performance for Autonomous Robot Missions with Uncertainty," *IFAC Int. Vehicle Symp.* 2013.

[10] D. MacKenzie and R. Arkin, "Evaluating the Usability of Robot Programming Toolsets," *Int. J. of Rob. Res.* 4(7) pp. 381-401 1998.

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

[12] M. O'Brien, R. Arkin, D. Harrington, D. Lyons and S. Jiang, "Automatic Verification of Autonomous Robot Missions," *4th Int. Conf. on Sim., Mod. & Prog. for Aut. Rob.* 2014.

[13] D. Lyons, R. Arkin, S. Jiang, T.-L. Liu and P. Nirmal, "Performance Verification for Behavior-based Robot Missions," *IEEE Trans. on Rob.* 31(3) 2015.

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

[15] S. Russel & P. Norvig, *Artificial Intelligence* Prentice-Hall, 2010.

[16] R. C. Arkin, *Behavior-Based Robots* MIT Press, 1998.

[17] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bull. Calcutta Math. Soc.* 35 pp. 99-109 1943.

[18] S. Livingston, R. Murray and J. Burdick, "Backtracking temporal logic synthesis for uncertain environments," *IEEE Int. Conf. on Rob. and Aut.* 2012.

[19] H. Kress-Gazit, T. Wongpiromsarn and U. Topcu, "Correct, Reactive Robot Control from Abstraction and Temporal Logic Specifications," *IEEE Rob. & Aut. Mag.* 18(3) 2011.

[20] M. Fisher, L. Dennis and M. Webster, "Verifying Autonomous Systems," *CACM,* 56(9) pp. 84-93 2013.

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

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

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