Verifying and Validating Multirobot Missions

D. M. Lyons *Senior Member, IEEE*, R. C. Arkin *Fellow, IEEE*, S. Jiang *Student Member, IEEE*, D. Harrington and T. Liu *Student Member, IEEE*

***Abstract*— We have developed an approach that can be used by mission designers to determine whether or not a performance guarantee for their mission software, when carried out under the uncertain conditions of a real-world environment, will hold within a threshold probability. In this paper we demonstrate its utility for verifying multirobot missions, in particular a bounding overwatch mission.**

I. Introduction

Deploying a team of autonomous or semi-autonomous robots to search for and locate a Chemical, Biological, Radiological, or Nuclear explosive threat provides the advantages of keeping human personnel remote from danger and also freeing the robots to make progress in the absence of regular supervisory communications. However, the extreme risk to the general population makes it incumbent on the designers of such missions to be able to give realistic performance guarantees of mission success. We have developed an approach that can be used by mission designers to determine whether or not a performance guarantee for their mission software, when carried out under the uncertain conditions of a real-world environment, will hold within a threshold probability. The paper builds upon our previous work in single robot missions [5][11].The contribution of this paper is the verification for multi-robot missions.

Our prior work, based on a process algebra methodology, addresses the verification of behavior-based robot programs without the discretization step seen in some work [7] and the computational complexity of a traditional model checking approach [6]. However, that work was single robot; The multirobot problem is more difficult since the robots can be interleaved in any order, introducing combinatorics which greatly complicate verification [18]. However, we show in Section IV that by leveraging the termination conditions of a recursive process definition our prior approach can be extended to handle multirobot mission verification. Our approach to verification does not include explicit search of all possible interleavings. We present the verification of a two robot, bounding-overwatch scenario using the new theoretical contribution. In addition to the verification of the performance guarantee for this example, we also present an experimental validation of this mission, and show that our prediction is consistent with the validation results.

II. Prior Work

Several methods have been developed to address the issue of general purpose software verification. Model-checking [6] is based on an exhaustive exploration of the states that can result from execution of the software. A critical issue in model checking approaches is potential state space explosion. Another approach is deductive verification [16] where properties are proved about the software using automated theorem proving techniques. An important issue to address for theorem proving techniques is how the assertions to be proved are generated.

Verification of robot mission software adds challenges to the general software verification problem. A robot program does not execute based on static inputs, but rather interacts with an asynchronous and concurrent environment model in an ongoing fashion. Because the end result of robot behavior involves motion and action in the physical world, there may be a necessary continuous nature to some aspects of the problem. The lack of full information about the nature of the environment in which a mission is to be carried out means that significant uncertainty must be modeled or verification results will not be realistic.

Brahman [3] uses model checking to automatically verify safety and system failure properties of programs specified with the MDS (Mission Data System) architecture. Using hybrid automata to address continuous spaces, she leveraged MDS to structure the problem to limit state explosion. Napp and Klavins [15] introduce a guarded command language CCL for programming and reasoning about multirobot. They address uncertainty by adding rates and exponential probability distributions to CCL, and address the multirobot issue by inspecting stochastic population properties.

Automatic controller generation [1] shares some of the characteristics of automatic verification. The starting point is a natural language description of the desired behavior [7] and the objective is to generate a correct-by-construction controller, typically exploiting model-checking techniques.

Multiagent systems (MAS) verification is addressed by Kouvaros et al.[18] in a model-checking framework. They address the combinatoric implications by restricting attention to systems where checking a (small) finite number of agents suffices to prove general properties. In contrast, we leverage the structure of behavior-based missions that are the input to our systems to address the combinatorics.

III. Verification System Architecture

Robotics has been considered as a key technology area to safeguard our society against attacks with weapons of mass destruction (WMD) [4]. However, due to the high-risk nature of counter-WMD (C-WMD) missions, failure cannot be tolerated. A robot team's success in a C-WMD mission has to be ensured before execution such that the robots would "get it

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

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

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

right the first time". We now review the framework that we developed to serve this purpose [12].

The verification framework is built upon *MissionLab*, a behavior-based robot programming environment [13], Figure 1. *MissionLab* is a user-friendly environment for designing multi-robot missions. The front-end of *MissionLab* is a usability-tested graphical robot behavior programming interface, where robot programs are created as finite state automata (FSA). The back-end of *MissionLab* provides a library of primitive behaviors (e.g., *GoToGoal*) that can be assembled into higher-level complex missions (e.g., a biohazard search behavior) in the form of FSAs.

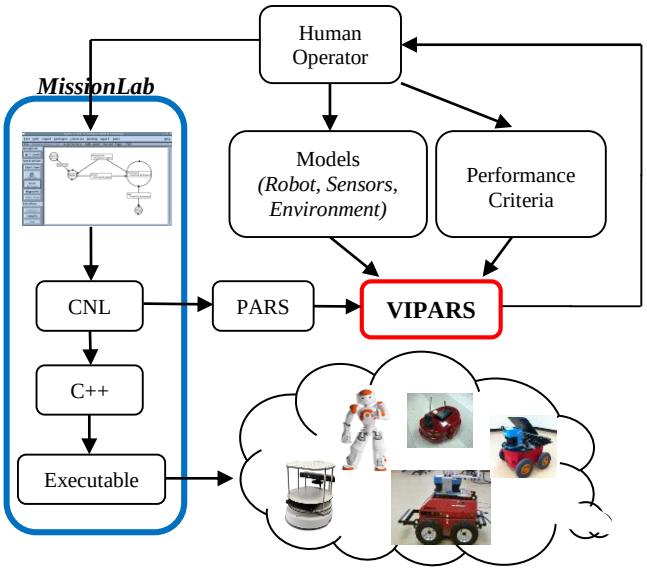


Figure 1: Verification System Architecture

The core of the verification framework is VIPARS (*Verification in Process Algebra for Robot Schemas*), a process algebra based approach to verifying robots' mission success. To initiate the verification of a multirobot mission, the robot programs in *MissionLab*'s CNL (*Configuration Network Language*) representation [14] are translated to PARS (*Process Algebra for Robot Schemas*), the language understood by VIPARS. The robot operator also needs to provide VIPARS with models of the robots, the sensors each robot is equipped with, and the environment the robots are to operate in, along with the performance criteria that the mission is required to meet. VIPARS then provides the operator with a performance guarantee for the mission based on how well the provided performance criteria were met.

By inserting VIPARS into the traditional program-execute loop of robot programming, it effectively forms a feedback design loop whereby the operator can iteratively refine the robot program based on information provided by VIPARS (e.g., time criterion is not met). Importantly, for critical missions such as C-WMD, operators can avert failures by not undertaking missions that are likely to fail.

IV. Formal Model

The *MissionLab* program, environment models, and performance criteria are all uniformly represented in PARS.

A. PARS

A brief overview of PARS is given here; for more details see [8]-[12]. The semantics of a process in PARS is a port-automaton extended with duration and partitioned end-states (separate abort and stop). A process **C** with initial parameter values *u1,u2,…*, in port connections *i1,i2,…*, out connections *o1,o2,…*, and final result values *v1,v2,…* is written:

$$\mathbf{C}\langle u1,u2,\ldots\rangle\,(i1,i2,\ldots)(o1,o2,\ldots)\langle v1,v2,\ldots\rangle \qquad (1)$$

The variables of the process are initialized by *u1,u2,…* The final result values *v1,v2,…*can be used to initialize other processes. The input and output connections indicate how **C** communicates with other, concurrent process.

Processes are either *basic* or *composite*, where a composite process is defined as a composition of other processes. A (conditional) sequential chain is composed using the ';' operator; a sequential chain will terminate once each process in turn stops, or if any process aborts. Other composition operations include parallel-max ('|') and parallel-min (cf. LOTOS disabling[2]) ('#') compositions. Further notation is explained as it is used in later sections.

B. CNL to PARS

In *MissionLab* a designer specifies the robot mission using as Finite State Automaton (FSA) (example is shown in Figure 5). Each state in the FSA involves the execution of a behavior which may result in many sensing and motor actions and interaction with the environment. Hence verification must occur at a greater level of the detail than the FSA.

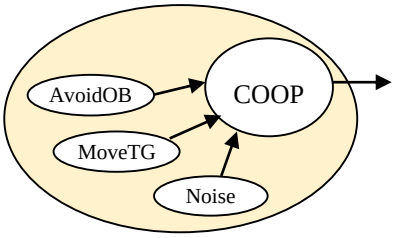


Figure 2:*GoToGuarded* Move CNL Network

Behavior states in the FSA are translated into more detailed networks of CNL processes with dataflow connections. For example, the *GoToGuarded* behavior is compiled into a network, as shown in Figure 2 (omitting lower-level data-flow nodes)which includes potential-field based obstacle avoidance.

A set of basic PARS processes has been defined to implement CNL nodes. These take similar parameters to the data-flow nodes and can be connected similarly. The *GoToGuarded* behavior is translated to PARS as a parallel composition of communicating processes as in Figure 2:

$$\begin{aligned}\mathbf{GoTG}\langle g\rangle(\mathrm{p,ob})(\mathrm{v}) = \;& \mathbf{MoveTG}\langle g\rangle(\mathrm{p})(\mathrm{c1}) \;\;|\\ & \mathbf{AvoidOb}\langle p\rangle(\mathrm{ob})(\mathrm{c2}) \;\;|\\ & \mathbf{Noise}\langle q\rangle()(\mathrm{c3}) \;\;|\\ & \mathbf{Coop}\langle w\rangle(\mathrm{c1,c2,c3})(\mathrm{v})\end{aligned}$$

The translation from CNL to PARS is currently done by hand using a template. An automated translator is being constructed. This topic is touched on again in Section VI.

C. System Description

The system to be verified consists of a robot program, translated from CNL to PARS, represented as a concurrent, communicating composition with the robot, sensor and environment models. A behavior-based program behaves dramatically differently in different environments, so robot, sensor and environment models are a crucial part of the system to be verified.

The robot program generates control commands for the robot, and queries information from physical sensors. The robot and sensor PARS models accept the same input commands and produce the same outputs through input and output ports. However, the relationship between the controller commands, the state of the environment and the sensor signals is determined by which models the user has chosen for verification. In prior work we have explored a number of models [8][9]; our focus here is on the problem of automated verification of multirobot missions given such models.

Consider a two robot program: the PARS program for the first robot is represented by the process **PR1** and for the second by **PR2.** The processes send control signals to and receive sensory information from their respective robot models **Robot1** and **Robot2** using ports. As a running example, consider the case where each robot receives position and generates velocity information:

$$\begin{aligned}\mathbf{Sys}\langle r,p,q,s\rangle = {} & \mathbf{PR1}\langle r\rangle\ (\mathrm{cp1})(\mathrm{cv1}) \ \ | \\ & \mathbf{PR2}\langle p\rangle\ (\mathrm{cp2})(\mathrm{cv2}) \ \ | \\ & \mathbf{Robot1}\langle q\rangle\ (\mathrm{cv1})(\mathrm{cp1}) \ \ | \\ & \mathbf{Robot2}\langle s\rangle\ (\mathrm{cv2})(\mathrm{cp2})\end{aligned} \qquad (2)$$

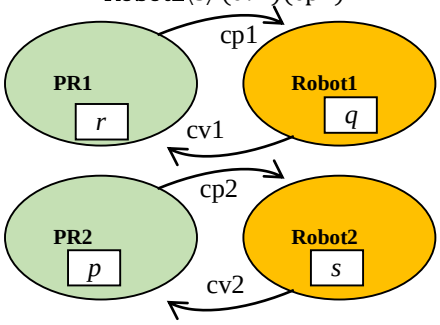


Figure 3: Two Robot Example System

Figure 3 is a graphical representation of (2), showing how the connections are made between robot program and robot model processes (the arrows) and also how the system parameters (*p,q,r,s*) relate to the parameters for each process. These parameters are a component of internal state for each process, where the initial position might be a parameter for the robot models.

In the case where all the processes in **Sys** are tail-recursive (TR) definitions (e.g. (3) below), then it may be possible to rewrite **Sys** itself as a TR process [10][9] where for convenience we denote all parameters with *u*:

$$\mathbf{Sys}\langle u\rangle = \mathbf{Sys'}\langle u\rangle; \mathbf{Sys}\langle f(u)\rangle \qquad (3)$$

The criterion developed in [10] for whether (3) is possible, is that the port operations in the TR body for each of the components of **Sys** can be matched up to produce a single TR body, **Sys'**. The advantage of (3) is that it enables efficient verification [9][11]. This operation has complexity $O(n_p n_{io})$ where n_p is the number of component processes and n_{io} is the number of port communication operations per process.

In one execution of **Sys'** the system parameters *u* are transformed to *f(u)* by calculations performed within component processes and by transfer of values by port communications between processes. Finding *f(u)* just requires following the connections between component processes [11]and has complexity $O(n_v n_p n_{io})$ where n_v is the number of system parameters.

D. Performance Criterion

The construction of the system period process, **Sys'** and the system flow function *f(u)*, are the prerequisites established in [10] for the verification of whether **Sys** satisfies a performance guarantee **G**, also specified as a process network. We argued that process algebra is just as intuitive a language to specify event orderings [2] as a temporal logic language (e.g., Linear Temporal Logic (LTL)). Since both program and specification are process networks, the bisimulation result relating property and program relates a more abstract specification of a process network to a more detailed implementation, as generalized by De Nicola [16]

Def. 1: P≈Q denotes process **P** and process **Q** are observationally equivalent by a specification implementation bisimulation.

Of course, in a logic it is possible to concisely proscribe, to state what should *not* hold, whereas a process algebra is prescriptive. We shall show that in our case this is not a limitation. Let us look at some examples.

A performance criterion network is written using process composition operations and basic processes along with constraints on the values of the parameters to the processes. An example of a simple liveness criterion is that robot *r1* reach a specific location; this is written as follows:

$$\mathbf{Delay}\langle t1\rangle\ ;\ (\mathbf{Delay}\langle t2\rangle\ \#\ \mathbf{At}\langle r1,p\rangle) \qquad (4)$$
$$t1 \le T,\ |p-L| < \varepsilon$$

The basic process **At**⟨*r,p*⟩represents robot *r* at position *p*. Specification (4) insists that *r1* arrive at location p by time T at the latest and the final position be within ε of location *L*. Notice there is no constraint on *t2*.

Example (5) specifies a two robot criterion. In this case, robot *r1* must arrive at its location before *r2* arrives at its location. Process variables in our framework may be *random variables* [11], and an example constraint on a random variable can be its probability of meeting some condition. In this case, the criterion specifies that the location of robot *r1* must have at least an *80%* probability of being within a distance *R* of location *L1* by time t*1*.

$$\begin{aligned}&\mathbf{Delay}\langle t1\rangle\ ;\ (\mathbf{Delay}\langle t2\rangle\ \#\ \mathbf{At}\langle r1,p\rangle)\ | \\ &\mathbf{Delay}\langle t3\rangle\ ;\ (\mathbf{Delay}\langle t4\rangle\ \#\ \mathbf{At}\langle r2,q\rangle) \\ &\qquad t1,t3 \le T,\ t1<t3, \\ &P\ (|p-L1|<R)>0.8,\ P\ (|q-L2|<R)>0.8\end{aligned} \qquad (5)$$

Finally, we consider a safety criterion: that the two robots never approach too closely. Notice this involves proscription, and it is handled by moving the negation to the constraint on the variable values. For all times greater than 0, the probability of the robots being ε must be at least 80%.

$$\begin{aligned}&\mathbf{Delay}\langle t1\rangle\ ;\ (\mathbf{Delay}\langle t2\rangle\ \#\ \mathbf{At}\langle r1,p\rangle)\ | \\ &\mathbf{Delay}\langle t3\rangle\ ;\ (\mathbf{Delay}\langle t4\rangle\ \#\ \mathbf{At}\langle r2,q\rangle) \\ &\qquad t1,t3>0,\ P\ (|p-q|>\varepsilon)>0.8\end{aligned} \qquad (6)$$

We define a performance criterion as follows:

Def. 2: *G*(**P**,*C*) is a performance criterion, where

- **P** is the process network associated with the criterion,
- *C* is the set of constraints on the process variables in **P**.

If *u* are the variables in **P**, then *C(u)* holds iff the constraints in *C* hold on the variable values of *u*. As usual, a safety criterion,(6) is treated as a negated liveness condition.

Def. 3: A performance criterion is negated by negating the constraint on variable values: $G(\mathbf{P},C) = G(\mathbf{P},\neg C)$

V. Verification of Missions

When **Sys** consists of only TR processes every computation of **Sys** can be expressed as a process network

(**Sys'**)^n and the values of the system variables are the solution of $f^n(u_o)$ where $f(u)$ is the system flow function, u_0 are the initial system parameter values and $n \geq 0$. Let **P** *sat* **Q** indicate that process **P** satisfies performance criterion **Q**, then:

Def. 4: Sys *satisfies* $G($**Q**$,C)$ iff, for $n \geq 0$

- (**Sys'**)^n *sat* **Q**, and $C(f^n(u_o))$

Def. 5: **P** *abs S* is the process in which any processes in **P** not named in the set *S* are hidden by the internal action **i**. This can be defined recursively:

- (**P** ; **Q**) *abs S* = (**P** ; **i**) if **Q** $\notin S$
- (**P** | **Q**) *abs S* = (**P** | **i**) if **Q** $\notin S$
- (**P** # **Q**) *abs S* = (**P** # **i**) if **Q** $\notin S$

For convenience, we also write **P** *abs* **Q** to mean the process where any processes not named in **Q** are hidden.

Def. 6: Compositions of internal actions can be grouped:

i ; **i** = **i**, **i** | **i** = **i** and **i** # **i** = **i**

Given a performance criterion $G($**Q**$,C)$ and **Sys** to be verified, then it is first necessary to evaluate (**Sys'**)^n *sat* **Q**. (See [9] for calculation of **Sys'**). This is accomplished in two steps: (**Sys'**)^n *abs* **Q** is calculated and hidden actions grouped, and then ((**Sys'**)^n *abs* **Q**)$\approx$**Q** is determined.

We have implement a simplified version of this procedure for speed: (**Sys'**)^n is projected to the set of processes in **Q** which we write (**Sys'**)$^n\downarrow$**Q**(or again (**Sys'**)$^n\downarrow$S where S is the set of processes in **Q**). This produces a version of (**Sys'**)^n *abs* **Q** in which all internal actions have been removed, and this is checked for *composition structural equivalence* $\approx_s$ with the performance criterion.

Def. 7: **P** $\approx_s$ **Q** holds iff

- **P**=**Q** or **P**= **P1** α **P2**, **Q** = **Q1** β **Q2** where, **P1** $\approx_s$**Q1**, **P2** $\approx_s$**Q2**, $\alpha = \beta \in \{';', '|', '\#'\}$

Structural equivalence is more restrictive (**P** $\approx_s$**Q** $\Rightarrow$ **P** $\approx$**Q** but not vice-versa) but is fast. Note that when compared structurally in this way, a program and specification may be equivalent on process names but differ on the parameters to the processes. For example, if we consider just the first part of the single robot move example in [9] we have the performance criterion:

Q = **Delay**$\langle t1 \rangle$; (**Delay**$\langle t2 \rangle$#**At**$\langle a \rangle$)

And after projection we have:

(**Sys'**)$^n\downarrow$**Q**= **Delay**$\langle (n\text{-}1)t \rangle$; (**Delay**$\langle t \rangle$ #**At**$\langle f^n(p) \rangle$)

In this case ((**Sys'**)$^n\downarrow$**Q**)$\approx_s$**Q** then the parameters to each process must be equal; i.e., $t1=(n\text{-}1)$t, $t2=t$ and $a = f^n(p)$, where t and p are constants. Thus, establishing **P** $\approx_s$**Q** posts a set of constraints on the parameter values which we write as $C_\approx$ which must also be met for the program to satisfy the performance criterion.

The final step in verification is to determine whether the constraints $C_\approx$ and C are satisfied by the system flow function: $C \cup C_\approx(f^n(u_o))$. This constraint problem can include random, continuous variables, and in [10] we map this to a filtering problem for a hybrid Dynamic Bayesian Network (DBN).

A. Non-TR Systems

In a multirobot, behavior-based system, we expect that the robot will continually respond to affordances in their environment. To handle this, we need to support *non-TR* processes in the concurrent communicating composition of the system process. In [12] we consider a waypoint mission, and in [5] a search and acquire C-WMD mission. Both of these require a single non-TR process in the system, and we extend our definition of *satisfy* to cover this.

Let **P** = **P1** ; **P2** ; **P3** ... **Pm** be the non-TR process in the system **Sys.** Let **Sys1** be **Sys,** where **P** is replaced by **P1**, and so forth for **Sys2** through **Sys***m***.** Let *f1* be the **Sys1** flow function, so forth for *f2* through *fm.* In that case:

Def. 8:Sys *satisfies* $G($**Q**$,C)$ iff for $n_1,n_2,..,n_m \geq 0$

- (**Sys1'**)^{n1}; (**Sys2'**)^{n2} ; .. ; (**Sys***m***'**)^{nm} *sat* **Q**, and
- $C(f1^{n1}(u_o)), f2^{n2}(u_1), fm^{nm}(u_{m-1}))$,

where $u_1 = f1^{n1}(u_o)$ and so forth for u_1 through u_{m-1}. This framework breaks the problem into a sequence of TR subsystems, addressing each one in turn to determine if it fulfills a portion of the performance criterion. For a waypoint example, the subsystems are the individual moves and the components of the criteria are the individual location constraints [12]. For a C-WMD search and acquire mission, the subsystems are the search component and the acquire component, and the components of the criterion are the target detection probability and final target location constraint [5].

However, a difficulty of multirobot systems is each agent will have one (or more) non-TR mission processes. Thus for k agents, this definition of satisfy would require k^m different sequences to be tested!

A TR process terminates when its body aborts (by the definition of ';'). *Basic condition processes* are basic processes that evaluate a condition on their arguments; aborting when their condition is true, stopping otherwise.

Def. 9: The abort condition of a process $\Omega($**P**$)$ is defined by:

$\Omega($**P**$) = cond($**P**$)$, if **P** $\in COND$,
$\Omega($**P** | **Q**$) = \Omega($**P**$) \wedge \Omega($**Q**$)$
$\Omega($**P** # **Q**$) = \Omega($**P**$) \vee \Omega($**Q**$)$
$\Omega($**P** ; **Q**$) = \Omega($**P**$)$; $\Omega($**Q**$)$

Where *COND* contains **EQ**$\langle$a,b$\rangle$, **NEQ**$\langle$a,b$\rangle$, **LT**$\langle$a,b$\rangle$, **GTR**$\langle$a,b$\rangle$ etc., and where *cond*(P) is the comparison condition for the basic process, e.g., *cond*(**EQ**$\langle$a,b$\rangle$) is *(a=b)* and so forth. Given the body of a TR process we can write the abort condition for the process in terms of the condition processes.

Let **Sys** be composed of k non-TR processes, each of which is m TR processes in sequence. We can assume sequential composition without loss of generality, since if at any point we have additional parallel or disabling components, they can be counted as additional non-TR components of **Sys**. We label the components $\mathbf{P}_{ij}$ for non-TR component $i \in \{1...k\}$, sequential component $j \in \{1...m\}$.

Our approach starts with **Sys1** containing $\mathbf{P}_{i1}$ and any other TR processes, $\mathbf{R}_1,\ldots,\mathbf{R}_h$. To know which components we need we evaluate (**Sys1**)^{n1} as before except that we use $\bigvee_i \Omega(\mathbf{P}_{i1})$ as a performance criterion. When (and if) this is satisfied, **Sys2** is then constructed from **Sys1** with any $\mathbf{P}_{i1}$ for which $\Omega(\mathbf{P}_{i1})$ holds replaced by $\mathbf{P}_{i2}$. We can now redefine *satisfy:*

Def. 10: Sys *satisfies* $G($**Q**$,C)$ iff for $n_1,n_2,..,n_m \geq 0$

- (**Sys1'**)^{n1}; (**Sys2'**)^{n2} ; .. ; (**Sys***m***'**)^{nm} *sat* **Q**, where
- **Sys1**=$(\mathbf{P}_{11}|\mathbf{P}_{21}|...\mathbf{P}_{k1})|(\mathbf{R}_1...\mathbf{R}_h)$ and **Sys1** *satisfies* $G(\mathbf{i}, \bigvee_i \Omega(\mathbf{P}_{i1}))$, and
- each subsequent system **Sys+** is the previous system **Sys-** with each $\mathbf{P}_{ij}$ for which $\Omega(\mathbf{P}_{ij})$ holds replaced by $\mathbf{P}_{il}$, $l=j+1$, and $C(f1^{n1}(u_o)), f2^{n2}(u_1), fm^{nm}(u_{m-1}))$.

Note: In the case of random variables, each $\Omega(\mathbf{P}_{ij})$ can be a

probabilistic condition.

VI. Verification & Validation of a Multi-Robot Mission

We now present the verification and validation of a *Bounding Overwatch* mission with two robots (Fig. 4), to illustrate the effectiveness of the VIPARS verification framework in providing performance guarantees for multi-robot missions. *Bounding Overwatch*, or *leapfrogging*, is a military tactical movement that is used by units of infantry to move forward under enemy fire. During this operation, one infantry unit does the *bounding* (i.e., advancing forward) while the other unit does the *overwatch* (i.e., stay stationary and watching over the *bounding* unit). The idea is that the *overwatch* unit is in a better position to watch for danger and to engage the enemy than the *bounding* unit [17].

Figure 4 presents a simplified *Bounding Overwatch* mission with two robots (designated as Robot1 and Robot2, Pioneer 3-ATs), where the robots alternate in bounding (i.e., advancing forward) and taking the overwatch position (i.e., covering for the advancing robot). The motivation behind this mission is to get the robot team to the end of the hallway, where an object of interest (e.g., biohazard) might be located, and to avoid detection by enemies by using *Bounding Overwatch*.

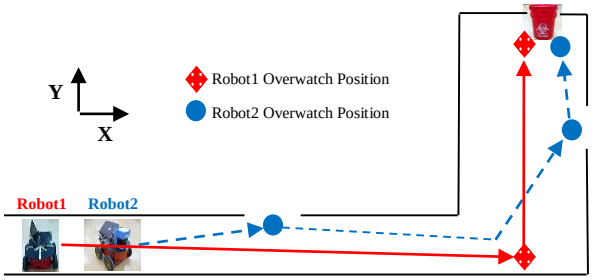


Figure 4: *Bounding Overwatch*

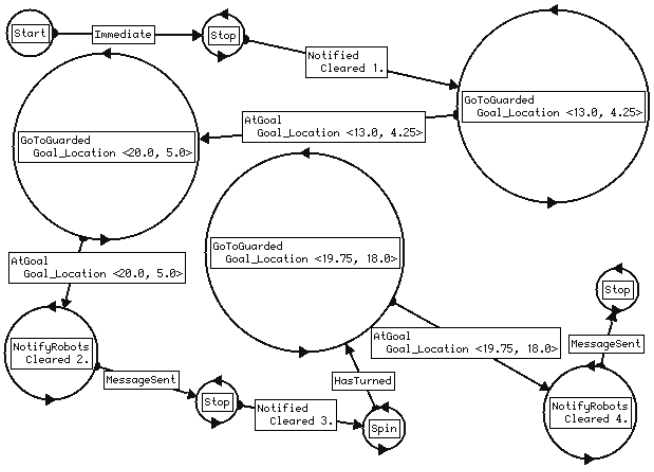


Figure 5: FSA for Robot1

The behaviors of Robot1 and Robot2 for carrying out the *Bounding Overwatch* mission are specified in *MissionLab* as FSAs; shown in Figure 5 for robot1. The FSA for Robot2 is similar to Robot1's and is not shown for brevity. The starting location (in meters) for Robot1 and Robot2 is (5.5, 5.0) and (6.5, 5.0) respectively; and the goal location for Robot1 and Robot2 is (19.75, 18.0) and (20.0, 17.5) respectively. Both robots use dead reckoning with shaft encoders and gyros to determine their positions. Each robot follows a series of waypoints, communicating a message to the other robot when it reaches its overwatch position. The mission consists of a set of behaviors (i.e., *GoToGuarded*, *Spin*, *Stop*, *Notify*) and triggers (i.e., *Notified*, *AtGoal*, *HasTurn*, *MessageSent*) for state transitions.

The performance criteria for the *Bounding Overwatch* mission are:

1. $\mathbf{R_{max}}$ – the success radius; each robot is required to be within this radius (e.g., 2.0 *m*) of its goal location in the physical environment
2. $\mathbf{T_{max}}$ – the maximum allowed time; the mission is required to be completed under this time limit (e.g., 200 *s*)

The *Bounding Overwatch* mission is only considered successful when both of these criteria are met. Thus, the overall mission success can then be defined as:

$$\textbf{Success} = (r_1 \leq R_{max}) \textbf{ and } (r_2 \leq R_{max}) \textbf{ and } (t \leq T_{max}) \quad (7)$$

Where r_1 and r_2 are Robot1's and Robot2's relative distances to their respective goal locations.

In the following subsections, we first describe the verification of the *Bounding Overwatch* mission with VIPARS. Then the details of validation of the mission with physical robots are presented. The results of verification and validation are then compared and discussed.

A. Verification

The CNL to PARS translation from the mission FSAs (e.g., Figure 5) yields a **Mission** process that includes the translated instructions for Robot1 and Robot2. A small portion of the **Mission** process is presented below:

```
1.Mission  (piM,piM2,obM,obM2,hM,hM2,odM,odM2)
2.           (vo,vo2) =
3.             ...
4. (Is_at_goal⟨G2B⟩(piM) ;
5.      Put_message⟨S1⟩(cG2))                              |
6. (Got_message⟨S1⟩(cG2) ; (
7.      Coop⟨1,1,1,G3A⟩(piM2,cV3,cC5,cC6)(vo2))            |
8.      Move_to⟨G3A,drop2500⟩(piM2)(cV3))                  |
9.      Noise⟨Z0,G3A⟩(piM2)(cC5))                          |
10.     Avoid_Obstacles⟨Z0,G3A⟩(piM2,obM2)(cC6)))          |
11. (Is_at_goal⟨G3A⟩(piM2);
12.     Spin⟨G90⟩(hM2)(vo2))                               |
13. (Is_at_head⟨G90⟩(hM2); (
14.     Coop⟨1,1,1,G3B⟩(piM2,cV4,cC7,cC8)(vo2))            |
15.     Move_to⟨G3B,drop2500⟩(piM2)(cV4))                  |
16.     Noise⟨Z0,G3B⟩(piM2)(cC7))                          |
17.     Avoid_Obstacles⟨Z0,G3B⟩(piM2,obM2)(cC8)))          | …
```

The Mission (controller) process inputs odometry and heading information while outputting a velocity for each **Robot** (environment) process; these are the port variables listed on lines 1 and 2. These carry information to and from the robot and sensor environment models. Trigger (basic) processes such as **Is_at_goal** and **Is_at_heading** (lines 4,6,11 and 13) are used to switch the behavior state by triggering process networks corresponding to the CNL implementations of the FSA states and connecting them to the **Mission** ports. The implementation of *GoToGuarded* in Fig. 2 can be seen at various points (lines 7-10 and 14-17).

The system process for this mission consists of the two robot models and the Mission process, similar to Figure 3 except with a single controller process communicating with each robot over different ports:

Sys = **Mission**(c1,c2,c3,c4, c5,c6,c7,c8)(c9,c10) |
Robot1⟨P01,Z0,H0⟩ (c9)(c7, c3, c5,c1) |
Robot2⟨P02,Z0,H0⟩(c10)(c8, c4,c6,c2) .

For example, *c9* and *c10* are the port connections for the velocity output for Robot1 and Robot2 respectively (see line 2 of **Mission**). The initial parameters to the robot processes fix the initial conditions for the verification.

The two robot models transform velocity commands to generate odometry (*q*) and actual position (*r*) according to the much simplified template below, where both *r* and *q* are random variables represented using a Mixture of Gaussians representation [11]:

Robot⟨r,q⟩ (v)(p,d) = (**At**⟨r⟩#**Delay**⟨t⟩
#**Out**⟨p,r⟩#**Out**⟨d,q⟩# **In**⟨v⟩⟨u⟩) ;
Robot⟨r+u*z(u,t)*t, q+u*t⟩.

Where *z(u,t)* is a function that calculates the uncertainty due to the velocity *u* applied for a discrete time step time *t* using the calibration data collected for each robot. Odometry can be read from port *d* and is calculated as dead reckoning. The odometry position is used by the Mission process to calculate its motions. The actual position *r* is calculated as a probability distribution using the uncertainty *z(u,t)*.

The performance criterion for the Mission from the prior section is effectively eq. (5) but without ordering constraint.

B. Validation

Experimental validation of the *Bounding Overwatch* mission, introduced in the beginning of this section, is conducted to show the validity of the VIPARS verification result. Validation consists of running the mission on physical robots in the real environment and measuring the robot team's performance with respect to the performance criteria R_{max} and T_{max}. The mission was run 100 times in an indoor lab environment with tile flooring, Figure 8. Both robots operated at a maximum velocity of 0.3 *m/s*; and they communicate with each other through the wireless network set up by a standard off-the-shelf router. The following three performance variables were measured during each trial run:

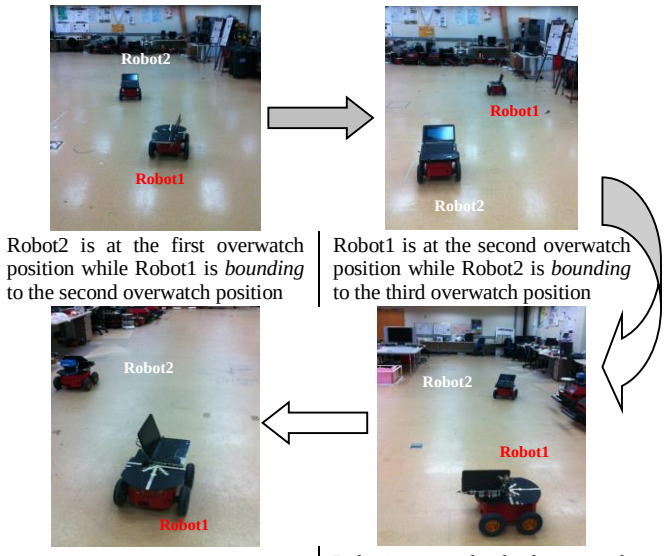
1. **t** – Mission completion time
2. $\mathbf{r}_1$ – Robot1's relative distance to its goal location
3. $\mathbf{r}_2$ – Robot2's relative distance to its goal location

C. Results

The goal of the verification framework is to serve as a decision support tool for human operators. While more research needs to be conducted to determine the best way to present performance guarantees to human operators under stressful situations, we present a preliminary representation. Instead of verifying the mission against a single specific point (i.e., a single (T_{max}, R_{max}) point), the mission is verified against a range of performance criteria values. We believe this would be more informative to the operator regarding the performance of the robot team in carrying out the operation. Furthermore, this could also demonstrate the robustness of VIPARS against a spectrum of performance criteria.

Figure 9 shows the VIPARS verification and experimental validation results for the performance guarantee of the time criteria $P(t \leq T_{max})$, the probability that the bounding overwatch mission will be completed under the time limit, T_{max}. Figure 9 shows that the VIPARS verification is consistent with validation result. The plot of verification versus validation of $P(t \leq T_{max})$, Figure 9 can be divided into three different regions:

1. *High Confidence (Unsuccessful)* – the region of near zero verification error and the mission succeeds with near zero probability, or *almost never* succeeds
2. *Uncertain* – the region where verification error is significantly greater than zero; verification error is defined as the difference between verification and validation
3. *High Confidence (Successful)* – the region of near zero verification error and the mission with success probability approaching 1.0 and *almost certainly* succeeds.



Robot2 is at the first overwatch position while Robot1 is *bounding* to the second overwatch position

Robot1 is at the second overwatch position while Robot2 is *bounding* to the third overwatch position

Robot1 is at its goal location while Robot2 is *bounding* to its goal location

Robot2 is near the third overwatch position; Robot1 is about turn 90 degrees counterlockwise and bound to its goal location

Figure 8: Snapshots of the *Bounding Overwatch* Mission

The *Unsuccessful* region informs the robot operator that if the performance criteria of the mission falls into this region, then the mission should be aborted or changed. For example, VIPARS verification of the time criterion, shown in Figure 9, informs the robot operator that the mission almost never succeeds under 180 *s*. Experimental validation of the time criterion, Figure 9, supports this interpretation (i.e., no validation run of the 100 trials completed the *Bounding Overwatch* mission under 180 *s*).

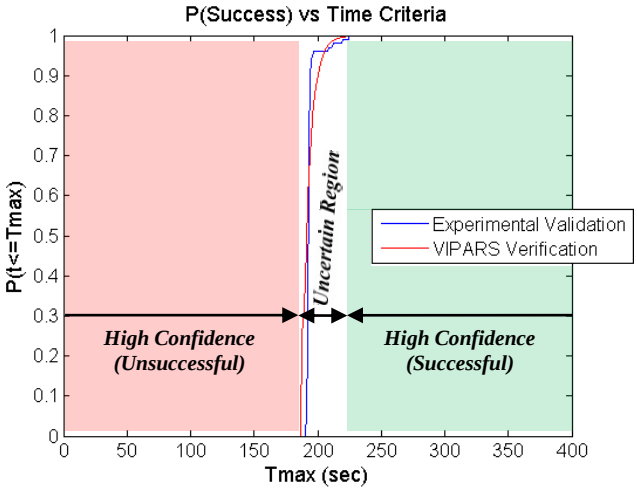


Figure 9: Verification vs. Validation of Time Performance Criterion $P(t \leq T_{max})$

The *Uncertain* region informs the robot operator that if the performance criteria falls into this region, then based on the acceptable threshold probability, she can decide to abort or execute the mission, although in general this area is to be avoided if possible. For example, if the mission is required to be completed under 200 *s* (i.e., T_{max} = 200), which VIPARS verified to have 0.9 probability of success. Then if 0.9 is greater than the threshold probability, the robot operator could proceed to execute the mission. From Figure 9, we also observe that higher the threshold probability (i.e., the minimum acceptable probability of success), higher the confidence of verification (i.e., the verification error decrease and gets closer to the *High Confidence (Successful)* region). Due to the prevalence of error between verification and validation in this region it should generally be avoided.

The *Successful* region indicates that if the performance criteria of the mission fall into this region, then the mission can be executed with high confidence that the robot team will *almost surely* get the mission right the first time. For example, Figure 9 tells the human operator that the robot team can complete the mission with close to 100% probability of success for $T_{max} \geq 225$ *s*. This is supported by the validation result, as no validation run of the 100 experimental runs of the mission completed under 225 *s*.

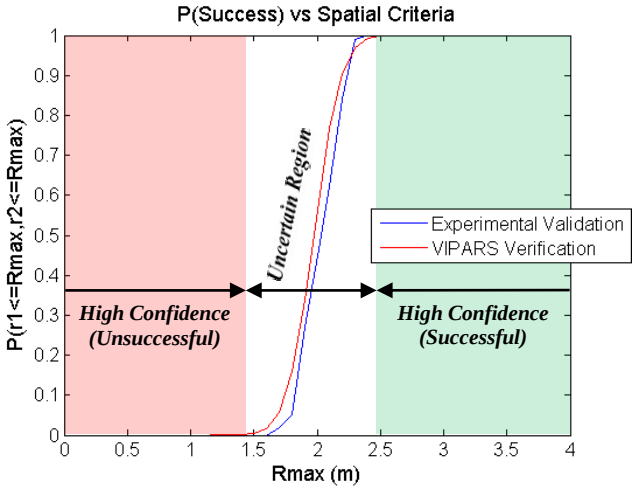


Figure 10: Verification vs. Validation of Spatial Performance Criterion $P(r_1 \leq R_{max}, r_2 \leq R_{max})$

Figure 10 shows the VIPARS verification and experimental validation results for the performance guarantee of the spatial criterion $P(r_1 \leq R_{max}, r_2 \leq R_{max})$, the probability that both robots will be within the R_{max} radius of their respective goal locations. Again, the VIPARS verification is consistent with the result of experimental validation as shown in Figure 10. Similarly, the result of the verification and validation of $P(r_1 \leq R_{max}, r_2 \leq R_{max})$ can be divided into three different regions as we have done for the time criterion above.

Figure 10 informs the robot operator to abort the mission if both robots of the *Bounding Overwatch* mission are required be within a radius of 1.5 *m* or less of their respective goal locations, since the mission *almost never* going to succeed in meeting the requirement. Additionally, depending on the acceptable threshold probability, the robot operator can make the decision to deploy or abort by evaluating the probability of success for R_{max} within the *uncertain* region in Figure 10. Lastly, the robot operator can infer that the mission almost surely will get it right for R_{max} greater than and equal to 2.5*m*.

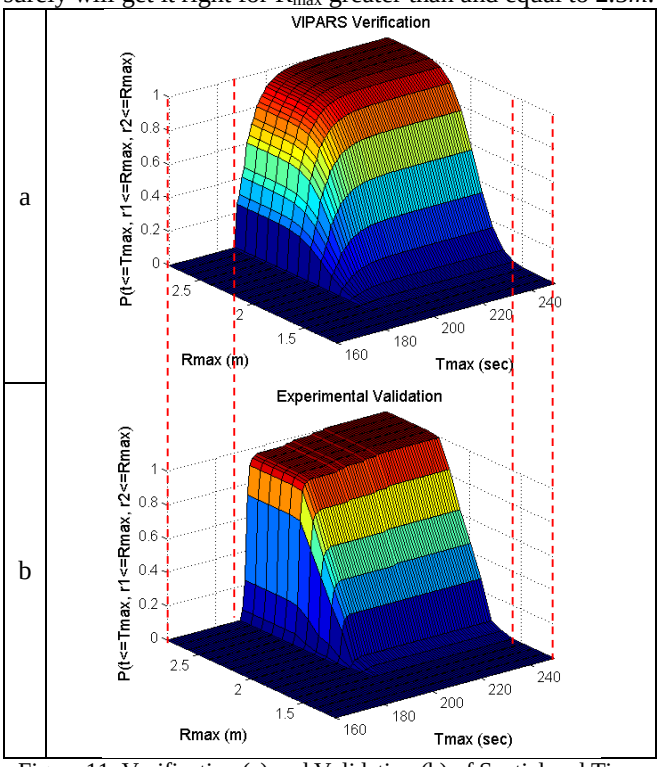
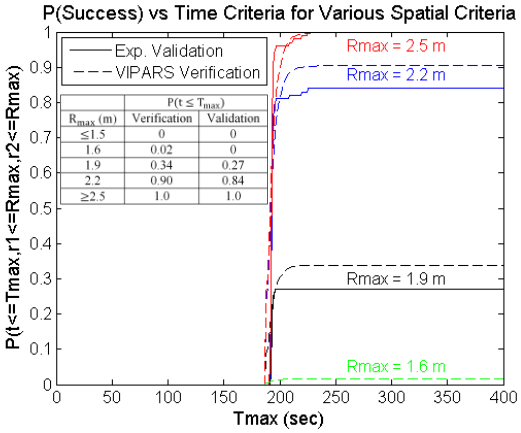


Figure 11: Verification (a) and Validation (b) of Spatial and Time Performance Criteria $\mathbf{P(t \leq T_{max}, r_1 \leq R_{max}, r_2 \leq R_{max})}$

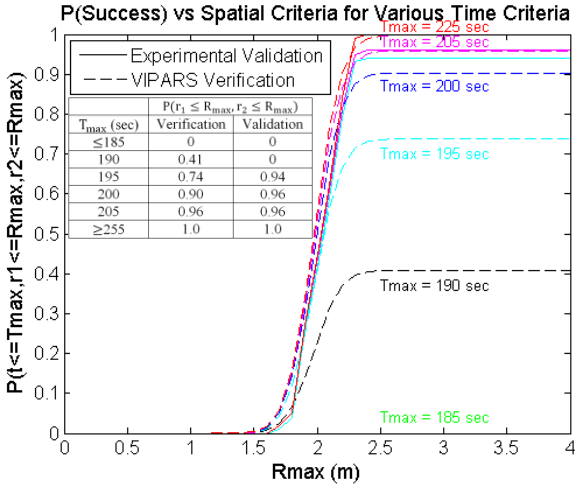
So far we have examined the two performance criteria separately. However, the *Bounding Overwatch* mission is only successful when both criteria are met. Figure 11 shows the VIPARS verification and experimental validation results for the performance guarantee of the overall mission success$P(t \leq T_{max} r_1 \leq R_{max}, r_2 \leq R_{max})$, the probability that both performance criteria, T_{max} and R_{max}, are met. The verification result is consistent with experimental validation. The dark red surface area in Figure 11 corresponds to the *High Confidence (Successful)* region while the dark blue surface area corresponds to the *High Confidence (Unsuccessful)* region. The result allows the operator to query the verification system for probability of overall mission success for different combination of performance criteria (e.g., spatial and time criteria for *Bounding Overwatch*).



R_{max} (m)	P(t ≤ T_{max}) Verification	Validation
≤1.5	0	0
1.6	0.02	0
1.9	0.34	0.27
2.2	0.90	0.84
≥2.5	1.0	1.0

Fig. 12: Verification & Validation of Time Criterion at various R_{max}

Figures 12-13 examine further how different combinations of performance criteria affect the overall mission success and verification error of VIPARS. These figures are basically 2D slices of the 3D surface plot in Figure 11. Figure 12 shows verification and validation of the time criterion at various spatial criteria. Comparing this to Figure 9, we see that while different values of R_{max} has no effect for the *High Confidence (Unsuccessful)* region, it significantly affects the probability of mission success in the *High Confidence (Successful)* region. More importantly, it also introduced significant verification error in this region. However, the values of R_{max} that significantly affects mission success and verification error are the R_{max}'s in the *Uncertain* region, Figure 10. This result reinforced our view that the *Uncertain* region should generally be avoided.



T_{max} (sec)	$P(r_1 \le R_{max}, r_2 \le R_{max})$ Verification	Validation
≤185	0	0
190	0.41	0
195	0.74	0.94
200	0.90	0.96
205	0.96	0.96
≥255	1.0	1.0

Fig.13: Verification & Validation of Spatial Criterion at various T_{max}

Fig. 13 shows verification and validation of the spatial criterion at various time criteria. T_{max} has a significant impact on the accuracy of verification in the *Uncertain* region. Figure 13 shows a large verification error for T_{max} = 190 *s*. On the surface, this error in the probability of overall mission success is due to the verification error of $P(t \le T_{max})$ at T_{max} =190 *s* in Figure 9 (at the beginning of the *Uncertain* region); however, the underlying culprit is likely the models of robot motion uncertainty that VIPARS used for verifying the performance criteria. Nonetheless, the time window for large verification is small (i.e., approximately 5 *s*); and the verification error decreases as T_{max} increases (e.g., T_{max} =195 *s*). Furthermore, VIPARS is more conservative than the validation result for T_{max} ≥195 *s* in the *Uncertain* region (e.g., VIPARS over-predicted the probability of success at T_{max} =190 *s*, and under-predicted at T_{max} =195 *s*). A conservative performance guarantee is desirable for critical multi-robot missions where failures could have catastrophic consequence.

VII. Conclusions

In this paper we have extended our approach to establishing performance guarantees for autonomous, behavior-based missions to handle the problem of multiple robot missions. Rather than exploring all possible action orderings for a mission expressed as a sequence tail-recursive (TR) system, each system is verified in sequence, and the end conditions used to determine which system to verify next. For systems with random variables this translates to looking at the maximum likelihood next action each time.

A bounding overwatch mission with two robots was verified and validated to illustrate the effectiveness of VIPARS in verifying multirobot missions. Rather than verifying one particular criterion point as in prior work, the mission is verified against a range of performance criteria values. We believe this would be more informative to the robot operator regarding the performance of the robot team in carrying out the operation. It also demonstrates the robustness of VIPARS against a spectrum of performance criteria. As prior work, verification results were validated against 100 trials of the mission on two Pioneer 3AT robots.

Probabilistic algorithms such as SLAM are a key part of modern robot software. Future work with VIPARS will involves performance guarantees for missions with probabilistic software.

References

[1] Belta, C. (2010) Synthesis of provably-correct control and communication strategies for distributed mobile systems. *ICRA'10 Workshop on Formal Methods for Rob.& Aut*.

[2] Bolognesi, T., and Brinksma,E. (1987) Introduction to the ISO Specification Language LOTOS, *Computer Networks & ISDN Sys*, 14(1), pp. 25-59.

[3] Braman, J. (2009) Safety Verification and Failure Analysis of Goal-Based Hybrid Control Systems. *Ph.D. Thesis.* Cal. Inst. of Technology, Pasadena CA.

[4] Doesburg, J.C. and Steiger, G.E. (2004) The Evolution of Chemical, Biological, Radiological, and Nuclear Defense and the Contributions of Army Research and Development, *NBC Report*, US Army Nuclear & Chemical Agency, Fall/Winter.

[5] Jiang, S., Arkin, R., Lyons, D., Liu, T-M., and Harrington, D. (2013) Performance Guarantees for C-WMD Missions. IEEE*Int. Symp. Safety, Sec.& Res. Rob.*, Linkoping Sweden.

[6] Jhala, R., Majumdar, R. (2009) Software Model Checking. *ACM Computing Surveys* 41(4) 21:1-53.

[7] Kress-Gazit, H., (2010) LTLMoP: Experimenting with Language, Temporal Logic and Robot Control.*ICRA'10 Workshop on Formal Methods for Robotics and Automation.*

[8] Lyons, D., Arkin, R. (2004) Towards Performance Guarantees for Emergent Behavior. *IEEE Int. Conf. on Rob. & Aut.*

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

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

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

[12] Lyons, D., Arkin, R., Nirmal, P and Jiang, Liu, T.M., S., Deeb, J. (2013b) Getting It Right The First Time: Robot Mission Guarantees in the Presence of Uncertainty. *IEEE/RSJ IROS*, Tokyo, Japan.

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

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

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

[16] De Nicola, R. (1987) Extensional Equivalences for Transition Systems, Acta Informatica, 24:211-237.Shankar, N. (2009) Automated deduction for Verification. *ACM Computing Surveys* 41(4) 20:1-56.

[17] Szczerba, R.,& Collier, B. (1998). Bounding overwatch operations for robotic and semi-robotic ground vehicles. SPIE Aerosense Conference on Guidance and Navigation.

[18] Kouvarous, P., Lomuscio, A. (2013) Automatic Verification of Parameterized Interleaved MultiAgent Systems. Proc. AAMAS13, St. Paul MN.