2017

Establishing A-Priori Performance Guarantees for Robot Missions that include Localization Software

Damian Lyons
Fordham University, dlyons@fordham.edu

Ron Arkin
Georgia Institute of Technology

Shu Jiang

Matt O'Brien

Feng Tang

See next page for additional authors

Recommended Citation

Damian Lyons, Ron Arkin, Shu Jiang, Matthew O'Brien, Feng Tang and Peng Tang, "Establishing A-Priori Performance Guarantees for Robot Missions that include Localization Software" International Journal of Monitoring and Surveillance Technologies Research (IJMSTR) Volume 5, Issue 1 2017.

Establishing A-Priori Performance Guarantees for Robot Missions that Include Localization Software

Damian Lyons, Department of Computer and Information Science, Fordham University, New York City, NY, USA
Ronald C Arkin, Georgia Institute of Technology, Atlanta, GA, USA
Shu Jiang, Georgia Institute of Technology, Atlanta, GA, USA
Matthew J O'Brien, Georgia Institute of Technology, Atlanta, GA, USA
Feng Tang, Fordham University, New York City, NY, USA
Peng Tang, Fordham University, New York City, NY, USA

ABSTRACT

One approach to determining whether an automated system is performing correctly is to monitor its performance, signaling when the performance is not acceptable; another approach is to automatically analyze the possible behaviors of the system a-priori and determine performance guarantees. Thea authors have applied this second approach to automatically derive performance guarantees for behavior-based, multi-robot critical mission software using an innovative approach to formal verification for robotic software. Localization and mapping algorithms can allow a robot to navigate well in an unknown environment. However, whether such algorithms enhance any specific robot mission is currently a matter for empirical validation. Several approaches to incorporating pre-existing software into the authors' probabilistic verification framework are presented, and one used to include Monte-Carlo based localization software. Verification and experimental validation results are discussed for real localization missions with this software, showing that the proposed approach accurately predicts performance.

KEYWORDS

Behavior-Based, Formal Verification, Localization, Robot Software, Uncertainty

1. INTRODUCTION

For systems that need to function in critical situations, such as in healthcare applications, search and rescue robotics, and automated counter weapons of mass destructions (CWMD) missions, it is crucially important that the system function as specified or the result might be loss of life, or property damage or both. One approach to this problem is to monitor the system in operation (Leucker & Schallhart, 2009) and to signal an alert to a supervisor of the system when the performance is going, or predicted to go, outside the necessary performance envelope. This monitoring approach can be very

DOI: 10.4018/IJMSTR.2017010103

effective in cases where a supervisor can step in, or the system can be disabled without consequence, when a performance problem is observed. However, in the case that the system is autonomous, or the supervisor cannot interact sufficiently quickly and the system cannot simply be disabled, then an alternate approach is necessary. In previous work for the Defense Threat Reduction Agency (Lyons et al., 2015) we have developed an efficient approach to the automatic, a-priori determination of performance guarantees for behavior-based robot missions operating in uncertain environments.

This work is related to formal software verification (Jhala & Majumdar, 2009) (DeMoura & Bjorner, 2012): a design tool to determine whether a piece of software will function properly without having to execute the software. The field has progressed strongly in recent years with developments in model-checking (Jhala & Majumdar, 2009) and satisfiability (SMT) engines (DeMoura & Bjorner, 2012). However, all such methods can at best approximate real robot performance because of the undecidability of the underlying verification problem. Designing a verification approach for mission critical robot software requires understanding what aspects of the robot software are of most importance to the problem. Behavior-based robot programming (Arkin, 1998) is an important tool in autonomous robotics that yields robot programs that are robust to uncertainty about exactly what environment the robots will face during execution. For this reason, in recent work (Lyons et al., 2015), we addressed the problem of automatically verifying behavior-based robot programs by leveraging the structure of such programs. The approach employs a unique combination of static analysis techniques and probabilistic reasoning to provide performance guarantees for behavior-based robot programs operating in physical environments with uncertain knowledge about obstacles to motion. Rather than addressing computational verification problems such as absence of deadlock or absence of run-time errors (Trojanek & Eder, 2014) (Walter, Taubig, & Luth, 2010), or verifying software generated control signals without consideration of the physical platform (Kim, Kang, & Lee, 2005), our work focuses on establishing performance guarantees for the mission software with a model of an uncertainly-known physical environment.

A key robotics development in recent years has been the use of probabilistic mapping and localization algorithms that allow a robot to operate robustly in previously unseen areas by automatically building a map and continually localizing the robot with respect to the map (Thrun, Burgard, & Fox, 2005). Many such algorithms have been programmed and tested and made available in software libraries (Dellaert, Fox, Burgard, & Thrun, 1999). Our prior work in verifying the performance of behavior-based robot missions involved manual and autotranslation of missions (O'Brien M., Arkin, Harrington, Lyons, & Jiang, 2014) specified using Georgia Tech's MissionLab (MacKenzie, Arkin, & Cameron, 1997) robot mission development toolkit. Other approaches to the verification of behavior-based systems have also been based on specific programming toolkits (Kiekbusch, Armbrust, & Berns, 2015). However, since there already exist software libraries for probabilistic mapping and localization, it would be advantageous to simply include one of these in a robot mission *if* it provided the necessary performance. In this paper, we address the problem of incorporating existing software into our novel approach to performance verification of behavior-based systems. Building on our work in (Lyons D., et al., 2016), we present some general theoretical results relating to this challenge, and then we address the specific problem of incorporating probabilistic localization (ROS AMCL) (Dellaert, Fox, Burgard, & Thrun, 1999) (Jiang & Arkin, 2015) into a C-WMD robot mission.

The remainder of the paper is laid out as follows. In Section 2, we review the relevant literature. In Section 3, we present as background an overview of our approach to verification. In Section 4, we present the main challenge: incorporation of existing software into the verification process, and we develop a theoretical basis for addressing this. Section 5 presents an application of the theory to the specific example of a C-WMD mission using the ROS AMCL localization module. Section 6 presents our conclusions.

2. LITERATURE REVIEW

Approaches to the problem of ensuring that a critical system will perform according to an a-priori formal performance guarantee include both run-time monitoring (Leucker & Schallhart, 2009) and formal verification (Jhala & Majumdar, 2009). Run-time monitoring has the advantage of typically being lightweight, however, it has the disadvantage that it can only be applied if a supervisor can be alerted, or the system disabled, in the event of a run-time monitor performance alert. Certain critical systems, such as health monitoring systems with life-critical intervention events, or autonomous systems handling explosives disposal with significant danger of life and property damage, do not have this property. In such cases, it is necessary to establish that the system will always abide by specified performance guarantees in advance. Our work addresses Counter-Weapons for Mass Destruction (C-WMD) robot mission software: software for multiple-robot teams searching for and locating WMDs. This falls into the category of software requiring a-priori performance guarantees.

Approaches to software verification include model checking (Jhala & Majumdar, 2009), static analysis (Venet, 2008) and theorem proving (Shankar, 2009). The model checking approach, which has proved widely successful, involves a comprehensive exploration of reachable states of the program to be verified. In the case of robot missions, it is very important to model the robot software, the properties of the robot actuators and sensors, and the behavior of the environment in which the robot is to operate. The addition of an environment model is rarely seen in general purpose software verification, but is crucial for any software that interacts with a complex physical system. Model checking approaches are sensitive to the size of the problem state space. This has been recognized as a problem in applying these techniques to robotics (Wongpiromsarn & Murray, 2008) because of the requirement to model sensors, actuators and environment behavior in addition to the software.

On approach to sidestep the additional state complexity (Fisher, Dennis, & Webster, 2013) is by verifying the robot's belief rather than the actual physical behavior. However, this begs the question of how belief will correspond to reality on any specific mission. We have followed an alternate approach (Lyons D., et al., 2013) (Lyons D., Arkin, Jiang, Harrington, & Liu, 2014) – avoiding making the state-space of the problem explicit. We developed a static analysis method to extract a set of flow-functions from behavior-based robot software. The functions describe how the variables of the program, robot, sensor and environment models recurrently interact during execution. The performance guarantee is mapped to boundary constraints on these functions. The flow functions which characterize the robot mission software are used to construct a Dynamic Bayesian Network (DBN), and verification of the software consists of filtering the DBN (as opposed to inspecting sample execution paths e.g., (Younes & Simmons, 2002)). Random variables are represented as Mixtures of Gaussians (MoG) (Lyons D., Arkin, Liu, Jiang, & Nirmal, 2013), essentially a multiple hypothesis representation for the variable value. Propagation of parametric distributions through a Bayesian network is very fast; many of the verifications take less than a few minutes despite extensive logging and debugging commands.

There are two challenges in automatically verifying existing probabilistic robot localization software. The first is that it requires an environment model, separate from, and interacting with, the robot software. The model has to include the physical location of the robot, the geometry of the map, and the relationship between these and the sensor measurements. Uncertainty in physical location (at the least) needs to be modeled. The second is that existing software must be directly included into the verification, rather than recoded and then included.

There is little directly related work in applying automated verification to critical missions in real-world environments. Proetzsch et al. (Proetzsch, Berns, Schuele, & Schneider, 2007) document verification of a behavior-based architecture of an outdoor mobile robot *RAVON*. They rewrite the software into the formal, synchronous language Quartz and carry out model-checking using the Averest model checker to ensure that the robot slows down and stops based on its proximity sensors. Since they have no environment model, what they verify is that the software issues appropriate velocity commands – in our work we verify the state of the robot itself. Walter et al. (Walter, Taubig, & Luth,

2010) also verify collision avoidance software for a mobile robot. They do include an environment model (their Domain model) and use the Isabelle theorem proving system. They manually annotate the code in their own formal language and use Isabelle as an assistant for human-driven, rather than automated, verification of the program.

Verification of software that includes probabilistic algorithms has been addressed by the formal verification community (Baeir & Katoen, 2008) (Katoen, 2010), and probabilistic model-checking tools such as PRISM (Kwiatkowska, Norman, & Parker, 2004) have existed for some time. However, such tools require a user to code their algorithm specifically for verification, whereas our objective is to have automatic verification, and to include existing probabilistic localization code into the verification process. The recoding approach has the general weakness that a re-implementation may not represent the actual code (it might be a better or worse implementation). Even published algorithm descriptions for widely known algorithms have been shown to contain errors (Zaks & Joshi, 2008). Recoding also inherently requires a large investment of expertise and manpower (Kim, Kang, & Lee, 2005). However, the main challenge in including preexisting code is that such code is designed to execute a single instance whereas verification reasons about all executions that are possible given the a-priori environment model information. Including existing code is possible in some model-checking tools – for example SPIN 4.0 and later allows C code to be embedded in a Promela model and the code is executed as an atomic transition within SPIN. Our approach is to consider the embedded code can transform a sample from a random variable, and we define a framework for sampling and reconstructing variable distributions.

3. VERIFICATION OF BEHAVIOR-BASED ROBOT MISSIONS

As background for our approach to addressing verification of existing localization code, we present an overview of our approach. The first subsection briefly introduces and motivates our theoretical approach using PARS (Process Algebra for Robot Schemas). The second subsection introduces the implementation in *MissionLab* and VIPARS (Verification in PARS).

3.1. Theoretical Framework for Automatic Verification of Behavior-Based Systems

As we have discussed, state-space explosion is a key challenge for automatic software verification. This problem is exacerbated in our case by the need to include not just the robot software, but also a model of the environment in which the robot program will operate. On top of this, both the robot software and the environment model will need to support the representation of uncertainty, a *sine qua non* in real-world robotics. To address these serious computational complexity problems, we elected to 1) not focus on state-based representations but rather process-based representations, and 2) to leverage the structure of behavior-based programming (Arkin, 1998) – a successful, robust robot programming paradigm.

A behavior-based program and its environment are modeled in PARS as a set of interconnected, recurrent processes, where a process **P** is written as:

$$P_{u_1,\ldots,u_n}\left(i_1,\ldots,i_j\right)\left(o_1,\ldots,o_k\right)_{v_1,\ldots,v_m} \tag{3.1}$$

where $u_1,\ldots,u_n$ are the *initial* values for the *process variables*, $i_1,\ldots,i_j$ and $o_1,\ldots,o_k$ are *input and output port connections*, and $v_1,\ldots,v_m$ are *final result values* of the process variables. Processes compute result values from initial values, and this computation may be influenced by any communications that occur over port connections. By focusing on processes rather than states, we shift the computational complexity from state combinatorics to the generator of those states – the process description. So, of course, we need a good language for that description.

Processes can be defined as combinations of other processes using the following composition operators: parallel ('|'), disabling ('#') and sequential (';'). Bounded recursion is captured using tail-recursive process definitions, e.g.:

$$Px = Qxy \; ; Py \tag{3.2}$$

Here process **P** first activates process **Q** with input value *x*. **Q** delivers output value *y*, which is then used to recur **P**. A variable flow function f_P relates the values of variables at the start of each recursive step of **P** to those at the end. The flow-function for atomic processes are specified *a-priori*; those for composite process, those defined as compositions of other processes, e.g., (3.2), are composed from the flow functions of the component processes. This can be automated in a standard static analysis (the approach used by compilers to check or optimize software) approach to generate flow functions given a set of processes (Lyons, Arkin, Jiang, Liu, & Nirmal, 2015) with complexity linear in the number of processes. Since any execution of Equation (3.2) is modeled by $f_P^{\,n}(x_0)$ for $n \geq 1$ and initial parameter value x_0, we have a straight-forward verification method. However, not all processes are defined in this form.

The system to be verified is expressed as the parallel, communicating composition **Sys** of a robot controller, e.g., **Ctr** with variable *r1,* and an environment model, e.g., **Env** with variable *r2,* written:

$$Sys\left\langle r_1, r_2 \right\rangle = Ctr\left\langle r_1 \right\rangle(a)(b) \middle| Env\left\langle r_2 \right\rangle(b)(a) \tag{3.3}$$

$$= Sys'\left\langle r_1, r_2 \right\rangle ; Sys\left\langle f_{Sys}\left(r_1, r_2\right)\right\rangle \tag{3.4}$$

$$f_{Sys}\left(r_1, r_2\right) = \left(f_{Sys,r1}\left(r_1, r_2\right), f_{Sys,r2}\left(r_1, r_2\right)\right) \tag{3.5}$$

In Equation (3.3), the input of **Ctr** is connected to the output of **Env**, (*a*), and the output of **Env** is connected to the input of **Ctr**, (*b*). If (3.3) were a sequential composition like (3.2) then we could extract flow functions for the combined interaction of controller and environment and conduct an efficient verification. Therefore, in (Lyons, Arkin, Jiang, Liu, & Nirmal, 2015) we developed *an* interleaving theorem[1] for behavior-based systems to convert Equation (3.3) to a sequential form Equation (3.4). The intuition is that a behavior-based system has behavioral 'states' each with an associated set of sensory-triggered responses. A static analysis algorithm *Sysgen* was developed to identify the set of processes for these states and rewrite parallel compositions of the form (3.3) into a sequential composition (3.4) where **Sys'** is the automatically identified behavioral state process, referred to as the system period. Once Sysgen analysis is complete, a system flow function can be extracted from **Sys'** in the usual way. In the small example of Equations (3.3), (3.4) above, the function extracted is shown in Equation (3.5). This is a recurrent function that evaluates the values for *r1* and *r2* as computed by the interactions between **Ctr** and **Env** in each execution of the system period **Sys'**. The result of this is that a parallel composition of controller software and environment model, such as in Equation (3.3), can be automatically verified by checking $f_{Sys}^{\,n}(x_0)$ for $n \geq 1$ and initial parameter value(s) x_0.

While this addresses the state combinatorics problem mentioned in the first paragraph of 3.1, it does not address the challenge of representing uncertainty mentioned in that same paragraph. Process variables, e.g., *r1, r2* in Equation (3.3), can be random or deterministic. Random variables are represented as multivariate mixtures of Gaussians, and operations on random variables are automatically translated into operations on distributions (Lyons D., Arkin, Liu, Jiang, & Nirmal, 2013); for example, a sum of random variables is translated to a convolution of their distributions. All these operations can be written (as approximations in some cases) as operations on the parameters of the Gaussian mixtures. Flow functions relate variable values at recursion step *t* of **Sys'** to those at *t+1*, and can be written as conditional probabilities, e.g.:

$$f_{Sys,r1}(r_{1,t}, r_{2,t}) = P(r_{1,t+1} | r_{1,t}, r_{2,t}) \quad (3.6)$$

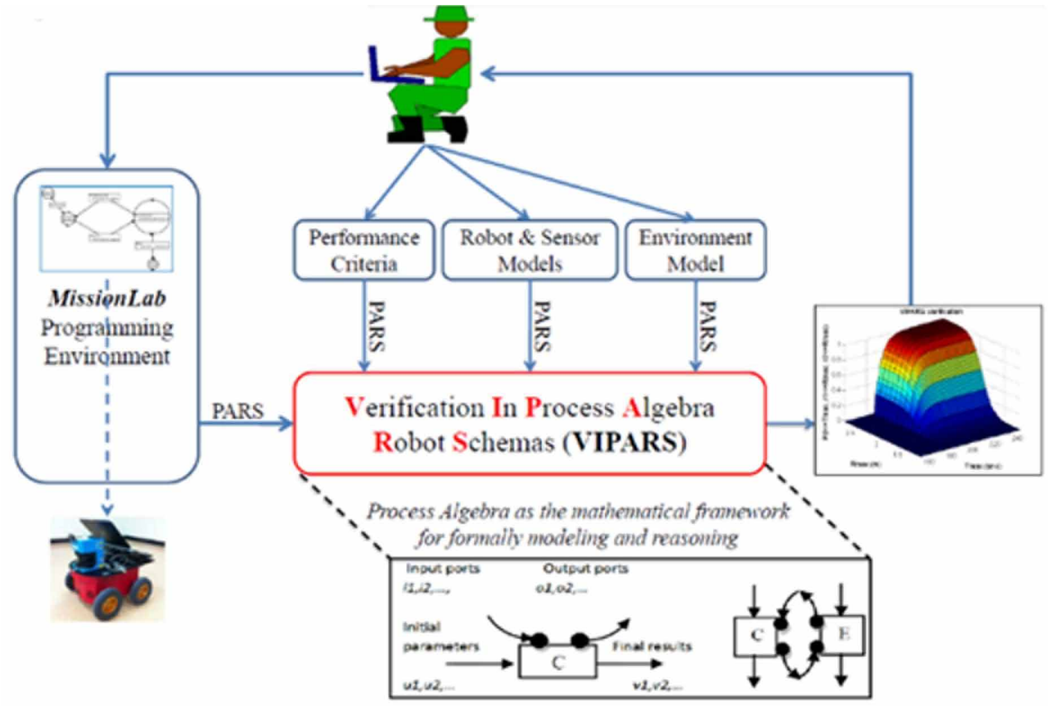
In the final phase of verification, extracted flow functions are converted to conditional probabilities e.g., (3.6). These are used to build a Dynamic Bayesian Network (Russel & Norvig, 2010) to carry out a forward propagation of probability distributions – a probabilistic version of $f_{Sys}^{n}(x_0)$ – to determine whether the combination of controller and environment will meet a performance specification. Although (Lyons, Arkin, Jiang, Liu, & Nirmal, 2015) discusses more complicated performance guarantees, we basically restrict our attention to the guarantee that a mission will achieve some criterion on environment variables (usually a spatial accuracy for a waypoint goal and/or a temporal requirement for achieving the mission) with probability greater than a threshold before a time-limit has expired. We have demonstrated that this approach is fast and accurate when validated against physical executions (most recently (Lyons et al., 2015)).

3.2. Implementation of Verification in MissionLab and VIPARS

The result of our research effort is a software verification tool VIPARS, which provides the performance guarantee for a robot mission based on how well the specified performance criteria are satisfied by the control program, robot, and the environment models for the mission.

VIPARS (Lyons D., Arkin, Jiang, Liu, & Nirmal, 2015) is built upon MissionLab (MacKenzie, Arkin, & Cameron, 1997), a behavior-based robot mission specification environment (Figure 1). MissionLab provides a usability-tested graphical programming interface, where the robot's program is specified in the form of a finite state automaton (FSA) assembled from a library of primitive behaviors. MissionLab missions are autotranslated (O'Brien, Arkin, Harrington, Lyons, & Jiang, 2014) to the process-algebra notation *PARS* for analysis. Environment models are also processes in this notation and we have proposed that a standardized set of environment models could be used

Figure 1. System architecture (reproduced from (Lyons et al., 2015))


Performance Criteria
Robot & Sensor Models
Environment Model
PARS
PARS
PARS
MissionLab Programming Environment
PARS
Verification In Process Algebra Robot Schemas (VIPARS)
Process Algebra as the mathematical framework for formally modeling and reasoning
Input ports
i1,i2,...
Output ports
o1,o2,...
Initial parameters
u1,u2,...
C
Final results
v1,v2,...
C
E

to capture different classes of environment (e.g., motion uncertainty (Lyons, Arkin, Jiang, Liu, & Nirmal, 2015); obstacle uncertainty (Lyons et al., 2015)).

The first phase of VIPARS verification is a static analysis of the concurrent, communicating mission software and environment model to generate a set of recurrent flow-functions capturing how the two interact. The second phase of VIPARS uses a Bayesian network to predict the performance of the mission from these functions.

The output of VIPARS is the performance guarantee, currently quantified as a set of probability distributions, that describes the likelihood of mission success. Rather than a single yes/no answer for whether the robot software being analyzed meets the performance requirements, VIPARS can output random variable distributions (e.g., the location of the robot at the end of the mission) and a set of performance graphs (e.g., the probability that a mission will be successful versus time, or the spatial accuracy of the final position as a graph of probability versus spatial error). This rich output fuels a feedback loop that allows the mission operator to improve the performance of the mission software.

4. VERIFYING EXISTING LOCALIZATION SOFTWARE

There are two key challenges to extending automatic verification to handle the verification of previously existing localization software incorporated into a robot mission. The first challenge is to provide an environment model for verification that includes the geometric and position uncertainty needed to verify any probabilistic localization approach. That will be addressed first, in subsection 4.1. The second challenge is to provide a framework for including the automatic analysis of previously existing software into probabilistic verification without requiring that the software be rewritten, annotated, or modified in any way. That will be addressed in subsection 4.2.

4.1. Environment Model for Localization

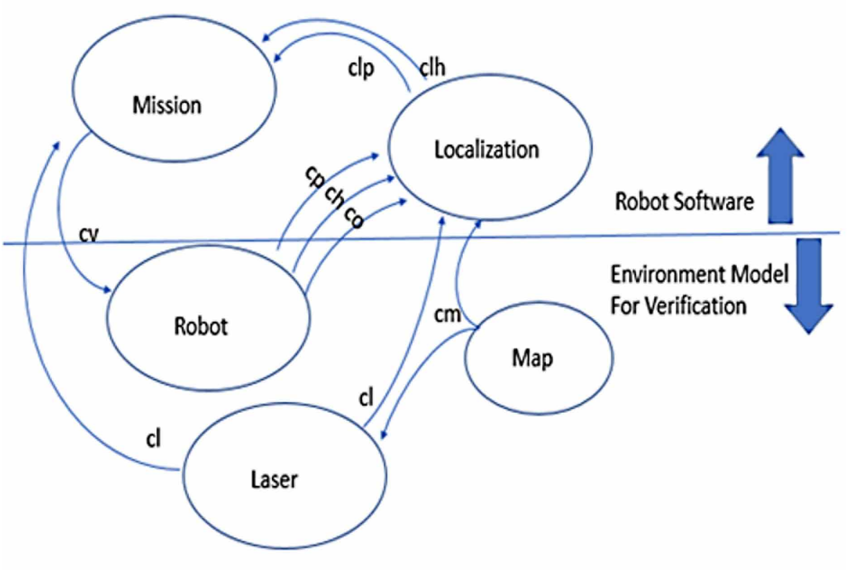
Section 3 included an introduction to the VIPARS verification module and to PARS, the process algebra framework underlying it. The software for a robot mission is translated from the MissionLab GUI Editor to a set of PARS process definitions. We refer to the top-level process as **Mission** and we verify this process by analyzing its interactions through the sensor and actuator signals it generates with a model of the environment in which it is to execute. This environment model is also defined as a set of PARS processes. The concurrent, communicating combination of the robot control software **Mission** and the environment model is called the system process, **Sys**.

In prior work, we have defined environment models to capture the motion and sensing uncertainty of the robot (Lyons D., Arkin, Jiang, Liu, & Nirmal, 2015), and environment obstacles with uncertain location and size (Lyons D., et al., 2015). To verify localization software, the environment model needs to contain information about the uncertain geometry of the environment. The system process **Sys** for the localization mission is shown in Equation (4.1) and illustrated as a process network in Figure 2:

$$\begin{aligned}Sys = \Big(& Mission\left(clp, clh, cl\right)\left(cv\right)\Big|Localization\left\langle D0\right\rangle\left(cp, co, ch, cl, cm\right)\left(clp, clh\right)\Big| \\ & Map\left\langle sysmap\right\rangle()\left(cm\right)\Big|MB\_Laser\left\langle ms, mo, lo\right\rangle\left(cm, cp, ch\right)\left(cl\right)\Big| \\ & Robot\left\langle P0, H0\right\rangle\left(cv\right)\left(cp, ch, co\right)\Big)\end{aligned} \tag{4.1}$$

Recall from Section 3, that the names in parenthesis after the process name are port connection names. The initial values for the process variables are show between angle brackets. The **Mission** software, as we'll see in a later section, programs a waypoint mission and is fundamentally similar to all prior waypoint missions we have verified and validated. It has inputs *clp* (position), *clh* (heading) and *cl* (laser readings); and output *cv* (velocity). **Robot** is the environment model, capturing motion and odometry error as well as the robot interactions with obstacles. It is also fundamentally similar

Figure 2. The system process Sys for the localization mission



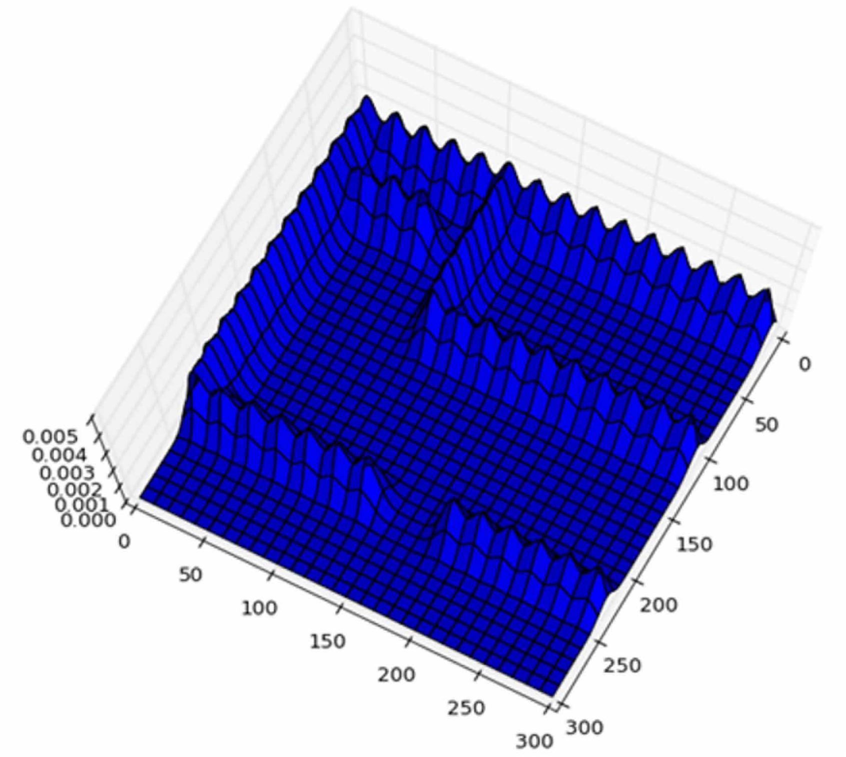
to the models used in our prior work. *PO, HO* are the initial robot position and heading, and the process ports are inputs *cv* (velocity) and outputs *cp, ch* (odometry position and heading) and *co* (real position distribution, i.e., without sensing noise – used for performance guarantee evaluation only).

In the behavior-based localization approach (Jiang & Arkin, 2015), the obstacle avoidance sensor (**MB_Laser**) gets its information from the map, rather than directly from measuring sensory input. **Map** makes map information available on its output *cm*; **MB_Laser** uses the map to generate map-based laser data on its output *cl*. **Localization** implements a localization method using the map *cm*, laser *cl*, and robot *cp, co, ch* inputs, where *D0* is the initial position uncertainty. The output of **Localization**, *clp*, is the localized position (and heading *clh*) used by the **Mission** process. How the **Localization** process encapsulates an existing localization algorithm will be addressed in subsection 4.2.

A key difference between this localization mission and prior missions to which we have applied our verification approach (Lyons D., Arkin, Jiang, Liu, & Nirmal, 2015) (Lyons D., et al., 2015) (O'Brien M., Arkin, Harrington, Lyons, & Jiang, 2014) is the map and the role it plays in both the obstacle avoidance behavior and in localization. The **Map** process in Equation (4.1) contains a map (*sysmap)* which represents the geometry of the physical environment in which the mission is executed. Our assumption is that this map has been generated a-priori for an environment of interest. Our concern here is how this map can be represented for the purposes of probabilistic verification.

Random variables are represented in VIPARS as Mixtures of Gaussians distributions (MG). If $a \sim MG(CM)$, for $CM=\{(\mu_i, \Sigma_i, w_i) \mid i \in 1\ldots m\}$ the set of the mixture parameters (means, variances, weights), then a_i refers to mixture member $N(\mu_i, \Sigma_i)$, and $w(a_i)=w_i$ are the mixture weights, where $\sum_{i=1}^{m} w_i = 1$, and $MG(x;\ CM)= \sum_{i=1}^{m} w_i N\left(x; \mu_i, \Sigma_i\right)$. The mixture size is written $\mid a \mid = m$. Map information – the locations and geometry of obstacles, walls and other physical aspects of the mission environment – can be directly represented using this mixture model as we explain below. The advantage of this approach to representing physical geometry is that there is no restriction on the spatial location or extent of obstacles, and finer precision of modeling can be obtained at the cost of adding more mixture members (Figure 3).

Figure 3. Example of VIPARS Gaussian mixture model map representation



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

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

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

During verification, the location random variable (the connection *cp* in Equation (4.1)) represents the location of the robot for *all* possible executions. It's relevant to compare this with the representation of robot location in a localization algorithm: the representation there may be also be a random variable, but the interpretation is different. In any single execution, the robot can only be at a *single* physical location; the localization distribution is an estimate of this. In verification, the objective is not to find the single most likely location, but to propagate the effects of being at all locations. Rather than using a ray trace algorithm to determine how each location is supported by sensor readings and refining the position estimate based on that, the ray trace algorithm is used by the MB_Laser process to gather *all* possible sensor readings that can arise due to the robot location distribution.

We begin by defining a ray scan on a map represented as an indexed mixture:

Definition: *Scanline*$((x_s,y_s), \theta) \subseteq R^2 = \{ (x,y): x=x_s + t \cos\theta, y = y_s + t \sin\theta, t \in R^+ \}$

The set of map mixture members encountered by *Scanline* is easily calculated as *Scanline* $\cap$ *I* and for convenience we also define the members encountered by *Scanline* on the indexed mixture *Map* as:

Definition: *Scanmix*$((x_s,y_s), \theta) \subseteq$ *Map* $=\{ (a[x,y], w[x,y]/\Sigma w[x,y]): a[x,y]\in Map \wedge (x,y) \in$ Scanline$((x_s,y_s),\theta) \cap Y \}$

The return from a sensor will then be an indexed univariate mixture calculated as follows. Let b_i for $i=1\ldots m$ be the members of *Scanmix,* then we calculate each member s_i of the sensor return *Sensor*$((x_s,y_s), \theta)$ from the members of *Scanmix*. Each member mean is the distance from the sensor origin to the map index and each member variance is the map accuracy variance convolved with a sensor specific noise distribution S_N:

$$s_i = s[\mu_i] = N(\mu_i,\sigma_i) * S_N \quad where \quad \mu_i = \left|(x_s,y_s)\,\mu(b_i)\right|, \;\; \sigma_i = \sigma_m \tag{4.2}$$

However, calculating the mixture weights is a little more involved due to the potential occlusion of mixtures (representing objects) by other mixtures (representing objects) that are closer to the sensor. The weights of the members of *Scanmix* need to be modified for the corresponding members of *Sensor* in order of visibility – that is, in order of μ_i as follows, where *w* are weights in *Scanmix* and *w'* the new, corresponding weights in *Sensor*:

$$w'_0 = w[\mu_0]$$

$$w'_i = w[\mu_i]\left(1 - \sum_{j=0}^{i} w'_j\right) \tag{4.3}$$

Sensor$((x_s,y_s), \theta)$ defines the sensor return to have an uncertainty related to the accuracy of the map. However, it does not include any uncertainty associated with the robot position. To incorporate the uncertainty in robot position into the sensor return, we extend the definition as follows:

Definition: *Sensor*$(N((x_s,y_s),\Sigma), \theta) = \{(s_i, w'_i): i=1\ldots m\}$ as in Equation (4.2) but where Σ is a diagonal covariance $\Sigma = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}$ and $\sigma_i = \sigma_m + (\sigma_x \cos\theta + \sigma_y \cos\theta)$.

However, this definition assumes a diagonal variance Σ for the robot motion uncertainty. This is unrealistic and we need to relax the assumption if we are to handle realistic robot motion. Any covariance matrix can be interpreted geometrically as the product of rotation R_θ and scaling *S* matrices:

$$\Sigma = \begin{bmatrix} \sigma_x^2 & \sigma_x\sigma_y\rho \\ \sigma_x\sigma_y\rho & \sigma_y^2 \end{bmatrix} = (R_\phi S)(R_\phi S)^T \; where \; R_\theta = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} and\, S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \tag{4.4}$$

Using this relationship, the angle (ϕ) and diagonal variances (s_x^2, s_y^2) of a non-diagonal covariance matrix can be identified and the scan angle rotated accordingly.

Definition: $Sensor\left(N\left(\left(xs, ys\right), \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}\right), \right) = Sensor\left(N\left(\left(xs, ys\right), \begin{bmatrix} s_x^2 & 0 \\ 0 & s_y^2 \end{bmatrix}\right), -\phi\right)$

Having ensured that environments, maps and sensor readings can all be reasoned about using the random variable framework in PARS – necessary for both obstacle avoidance and localization – we next turn our attention to localization itself.

4.2. Including Existing Software in Probabilistic Verification

The main difficulty with the incorporation of existing localization code *directly* into the VIPARS verification algorithm is that localization code is designed to execute just a single instance of a robot mission, whereas VIPARS is probabilistically reasoning about all executions that are possible given the a-priori environment model information. Consider that the C++ program we want to add to a mission is **P**. A PARS process wrapper for **P** is built, so the code behaves like a 'black box' process $\mathbf{P}\langle x\rangle\langle y\rangle$. However, when **P** is called, it will map one input value *x* to an output, *y*; only one possible execution of **P**, whereas verification has to check *all* possible executions.

Our approach to this challenge considers building a process wrapper for the embedded code that is capable of transforming a sample (which represents a single instance of a mission) from a PARS random variable, and for this purpose we define a framework for sampling and reconstructing variable distributions. This approach has the advantage of using the actual code that will get executed by the robot at run-time for the mission. It has the disadvantage of potentially lengthening verification times, since multiple samples need to be evaluated for a representative result.

We define an extension to the flow function f_P for the process/program **P** (representing the actual embedded code): the mixture extended flow function F_P takes a random variable *x* as input and produces a random variable **y** as output. It samples the input distribution *x* and calls f_P on the samples, and reconstructs the output distribution mixture $p(y \mid x) = F_P(\boldsymbol{x})$ from the result. Let **x,y** ~ MG(CM); our objective is to evaluate $\mathbf{y} = \Pi(\mathbf{x})$ ($=f_P$ above) where Π is the transformation carried out by the embedded code on the variable x. We will present several ways this can be done.

Definition: The MG mean-extended real function (MER) **Π** is defined as follows:

- If Π is defined as $\Pi: \mathfrak{R} \rightarrow \mathfrak{R}$, where $y = \Pi(x)$, for $x, y \in \mathfrak{R}$, then;
- The MG mean extended real function **Π** is defined as $\mathbf{\Pi}: MG \rightarrow MG$;
- Where $\mathbf{y} = \mathbf{\Pi}(\mathbf{x})$, for $\mathbf{x},\mathbf{y} \in MG$ (and *MG* is the set of all MGs); and
- Where we define $\mathbf{y} = \mathbf{x}$;
- Except $\boldsymbol{\mu}(y_i) = \Pi(\boldsymbol{\mu}(x_i))$ for all x_i in **x**.

So, the MER function preserves number of members and variances. Only the mean is transformed, and a normal distribution is transformed to another normal distribution with the same variance. This limits our modelling of what the software in Π can do:

- It cannot for example add or reduce uncertainty since variance is unchanged. The entire distribution is moved to center over a new mean;
- It cannot add new members to the mixture since the number of members is unchanged.

This is clearly very limiting, though it might be sufficient for a fast estimate of performance. Consider that if we call Π with a single number *x*, it returns a normal distribution with the result for *x* as the mean and a variance: **y** = N(**μ**,**σ**) = Π(x). Now, if the input to code is a MG **x**, then we could define a MG extension of Π as:

Definition: The MG *mean extended normal function (MEN)* **Π** is defined as follows:

- If Π is defined as Π: 𝕽→𝕽×𝕽, where (**μ**,**σ**)=Π(*x*), for *x*, **μ**,**σ** ∈𝕽, then;
- The MG normal extended function **Π** is defined as Π: *MG* → *MG*;
- Where **y** = **Π**(**x**), for **x**,**y** ∈ *MG*; and
- Where we define **y**=**x**;
- Except (**μ**(y_i), **σ**(y_i)) = Π(**μ**(x_i)) for all x_i in **x**.

Notice however that the MEN function effectively ignores **σ**(x_i); only the mean is used. In probabilistic terms, this is like only operating on the most likely value. However, since a variance is returned, it means the code does include an increase or decrease in the uncertainty associated with the transformed variable.

Definition: The mean and variance of an MG is defined similarly to the mean and variance of a mixture of Gaussians. For a random variable **x** ~ MG:

- $\boldsymbol{\mu}(\mathbf{x}) = \sum_{i=1}^{m} w_i \ \mu_i,$ that is, the weighted sum of means;
- $\boldsymbol{\sigma}(\mathbf{x}) = \sum_{i=1}^{m} w_i \left(\left(\mu_i - \right)^2 + _i^2 \right)$

Let's consider the input x_i to actually be a collection of samples, where the weight of sample s_j is given by N(s_j ; μ_I, σ_i) – that is, member *i* evaluated at s_j. This discretizes and samples the a-priori distribution. We can evaluate the MEN function for each sample, and then use the results to reconstruct the a-posteriori (posterior) distribution as follow:

Definition: The MG *extended normal function* (ENF) **Π** is defined as follows:

- If Π is defined as Π: 𝕽→𝕽×𝕽, where (**μ**,**σ**) = Π(*x*), for *x*, **μ**,**σ** ∈𝕽, then;
- The MG extended normal function **Π** is defined as **Π**: *MG* → *MG*;
- Where **y** = **Π**(**x**), for **x**,**y** ∈ *MG*; and
- Where we define **y** = **x**;
- Except (**μ**(y_i), **σ**) = Π(**μ**(x_i)) for all x_i in **x**; and
- Where **σ**(y_i) *is calculated as follows*:
  - (**μ'**$_j$, **σ'**$_j$) = Π(s_i) for s_i a sample of the input x_i;
  - $\boldsymbol{\sigma}(y_i) = \sum_{j=1}^{k} N\left(s_j \ ; \left(\mathrm{x}_i\right), \left(\mathrm{x}_i\right)\right) \left(\left(\mu'_j - \left(\mathrm{y}_i\right) \right)^2 + '^2_j \right)$

So rather than evaluating the program Π on a single number, it is evaluated on a set of numbers of predefined size (*k*) and the results combined to form a single normal distribution.

It's also possible to take a sample approach to a real function Π, and calculate a variance on the result sample.

Definition: The *MG extended real function* (ERF) **Π** is defined as follows:

- If Π is defined as Π: 𝕽→𝕽, where y=Π(*x*), for *x*,*y* ∈𝕽, then;
- The MG mean extended real function **Π** is defined as Π: *MG* → *MG*;
- Where **y** = **Π**(**x**), for **x**,**y** ∈ *MG*); and

- Where we define $\mathbf{y} = \mathbf{x}$;
- Except $\boldsymbol{\mu}(y_i) = \Pi(\boldsymbol{\mu}(x_i))$ for all x_i in $\mathbf{x}$; and
- Where $\boldsymbol{\sigma}(y_i)$ *is calculated as follows*:
  - $\boldsymbol{\mu}'_j = \Pi(s_i)$ for s_i a sample of the input x_i;
  - $\boldsymbol{\sigma}(y_i) = \sum_{j=1}^{k} N\left(s_j;\left(x_i\right),\left(x_i\right)\right)\left(\left(\mu'_j - \left(y_i\right)\right)^2\right)$

This set of definitions for MEN, MER, ENF, and ERF, allows for a range of options in interfacing embedded code with verification in general, and with VIPARS in particular.

5. VERIFICATION AND VALIDATION OF A ROBOT MISSION WITH LOCALIZATION

To assess the effectiveness our approach in providing performance guarantees for probabilistic localization missions, we present two waypoint missions where the robot is tasked to navigate through a series of waypoints, avoiding obstacles and toward a goal with behaviors that are based on probabilistic localization. The general assessment process consists of three steps: 1) verification – use VIPARS to generate a performance guarantee for the mission with respect to some specified performance criteria, 2) validation – conduct experimental trials of the mission with a real robot, 3) evaluation – compare the predicted performance generated by VIPARS with the actual performance of the robot.

The waypoint missions are illustrated in Figure 4. The mission proceeds with robot starting at (2, 2) and navigates by following a series of waypoint to the goal locations at (11.7, 12.5) and (1.0, 7.3) respectively for each mission. The behavior of the robot for *Mission-B* (Figure 4a) is shown in Figure 5, which was created in *MissionLab* in the form of an FSA. The FSA consists of a series of *GoToGuarded* and *Spin* behaviors, whose transitions are prompted by *AtGoal* and *HasTurned* triggers. The FSA for *Mission-A* is similar to the one shown in Figure 5 but is omitted for brevity.

In contrast to the behaviors we had examined in our prior work, the behaviors here have leveraged a probabilistic localization algorithm to improve mission performance. Specifically, the perceptual schemas of *MoveToGuarded* and *AvoidObstacles*, two of the constituent primitive behaviors of the high-level *GoToGuarded* behavior, are augmented with a SLAM-based spatial map (Jiang & Arkin,

Figure 4. Waypoint missions for verification and validation

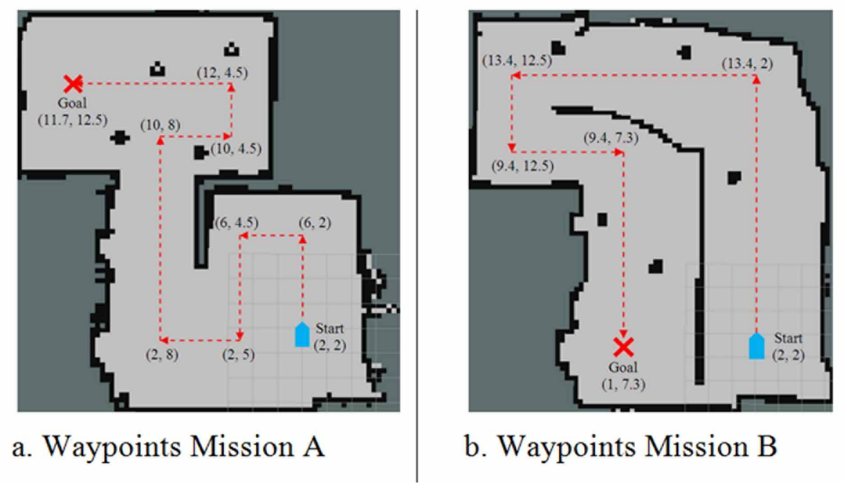
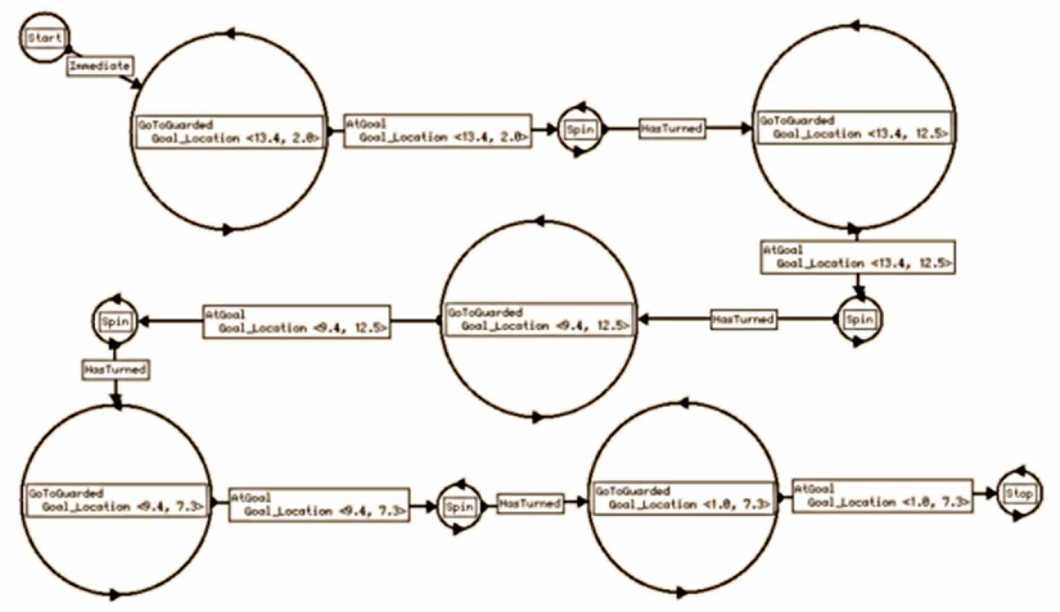


Figure 5. Behavioral FSA for Mission-B


Start
Immediate
GoToGuarded Goal_Location <13.4, 2.8>
AtGoal Goal_Location <13.4, 2.8>
Spin
HasTurned
GoToGuarded Goal_Location <13.4, 12.5>
AtGoal Goal_Location <13.4, 12.5>
Spin
HasTurned
GoToGuarded Goal_Location <9.4, 12.5>
AtGoal Goal_Location <9.4, 12.5>
Spin
HasTurned
GoToGuarded Goal_Location <9.4, 7.3>
AtGoal Goal_Location <9.4, 7.3>
Spin
HasTurned
GoToGuarded Goal_Location <1.8, 7.3>
AtGoal Goal_Location <1.8, 7.3>
Stop

2015). The *MoveToGuarded* primitive behavior drives the robot to a specified location with a radius of velocity dropoff around the goal. Instead of using odometry for localization, the perceptual schema of *MoveToGuarded* is replaced with the adaptive Monte Carlo localization (AMCL) algorithm (Dellaert, Fox, Burgard, & Thrun, 1999). This probabilistic localization algorithm takes the robot odometry and an a-priori acquired map as inputs, and outputs an estimated pose of the robot along with a covariance matrix representing the uncertainty of the estimated pose. Furthermore, the *AvoidObstacles* behavior uses the spatial map, instead of using direct sensory reading from the laser scanner, to generate repulsion vectors. The perceptual schema of the *AvoidObstacles* behavior is modified to turn the spatial map into pseudo laser scans of the environment through ray tracing within the occupancy map. As a result, the *GoToGuarded* behavior utilizes perceptual information (i.e., robot pose and obstacles) generated by probabilistic algorithms to produce motion commands while navigating through the waypoints.

Performance criteria are mission specifications that must be met. For missions A and B, these include constraints on:

- R_{max}: Maximum radius of spatial deviation allowed from the goal;
- T_{max}: Maximum allowable mission completion time.

In this example, each waypoint mission is considered successful only when two constraints are met:

$$\text{Success} = (r \leq R_{max}) \text{ and } (t \leq T_{max}) \tag{5.1}$$

where r is the robot's relative distance to its goal location and t is the time for the robot to finish a mission. The objective of VIPARS is then to verify how well these performance criteria are satisfied by the combination of the robot, its control software, and the operating environment.

5.1. Verification

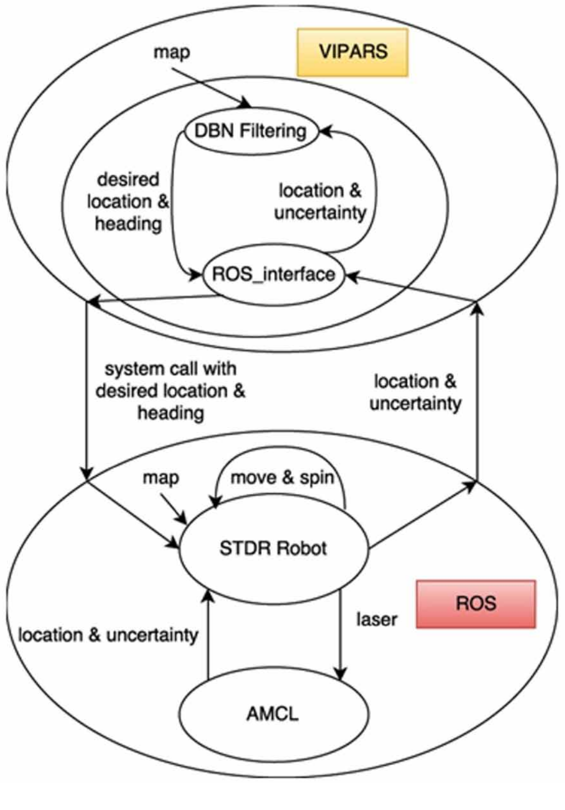
The localization algorithm used in this paper was Adaptive Monte Carlo Sampling (AMCL) (Fox, 2001) as implemented in ROS. In the sampling approach, the DBN filtering engine of VIPARS issued requests to a ROS-based AMCL server to evaluate the ERF function for the **Localization** process. The interaction is shown in Figure 6: Whenever the flow function for the **Localization** process needed to be evaluated on a position random variable, the position variable was sent from the DBN filtering engine (Top, Figure 6) via a pipe to a concurrently running ROS Indigo system (Bottom, Figure 6). The ROS STDR simulator node was instructed to move the robot to the appropriate position, and localization data collected from the AMCL node. For simplicity, the ERF function was restricted to single member mixtures, and rather than calculating the variance by evaluating multiple samples, only the mean value was transformed and the variance calculated by convolving the mean with a zero-mean distribution *N*$(0, \sigma_s)$. This simplified the hysteresis issue with calling AMCL. The hysteresis challenge in fully implementing the ERF Definition for AMCL is discussed in the Conclusion.

The results of carrying out verification on both waypoint missions was a set of performance graphs (as described Section 3) showing the predicted performance of the missions with respect to the performance criteria (5.1).

5.2. Validation

Validation experiments of the waypoint missions were conducted to illustrate that VIPARS' predicted performance of the mission is consistent with the robot's actual performance. The robot used for the experimental trials is the Pioneer 3-AT, a four-wheeled skid-steered mobile robot. The robot

Figure 6. VIPARS-ROS architecture



is also equipped with a forward-facing SICK laser scanner. The complete validation experiment consists of 50 trial runs for each waypoint mission respectively, which resulted in a total of 100 trial runs. Snapshots of the waypoint mission B are shown in Figure 7. Mission success is defined by how well the performance criteria in (5.1) are met. For each trial, the following performance variables were measured:

- ***t*****:** Mission completion time;
- ***r*****:** Robot's relative distance to its goal location.

5.3. Verification vs. Validation (V&V)

Verification and validation are conducted independently by our two research groups, and the results are not shared until the final comparison stage. Figure 8 shows the results of verification and validation of the waypoint missions. The performance guarantee is quantified as a probability distribution that represents the robot mission's likelihood for success. These results also serve as the basis for performance feedback; and how this information should ultimately be presented to the mission operator was investigated in our recent human-subjects study (O'Brien & Arkin, 2016).

Figure 8 compares the validation and verification results of the performance guarantees for the two waypoint missions. Figures 8a and 8c show the V&V results for the spatial criterion $P(r \leq R_{max})$ of (5.1), the probability that the robot arrives within R_{max} radius of its goal location. Figures 8b and 8d show the comparisons for the time criterion $P\left(t \leq T_{max}\right)$, the probability that the waypoint mission is completed under the time limit, T_{max}. The results illustrate that the VIPARS verification of performance guarantees are consistent with the outcomes from experimental validation.

The V&V results can be divided into three regions for further interpretation: High Confidence (Unsuccessful), Uncertain, and High Confidence (Successful) regions. The High Confidence (Unsuccessful) is the region of near zero verification error and the mission has a zero probability of success. The Uncertain region is the region where verification error is significantly greater than zero

Figure 7. Snapshots of validation for Mission-B

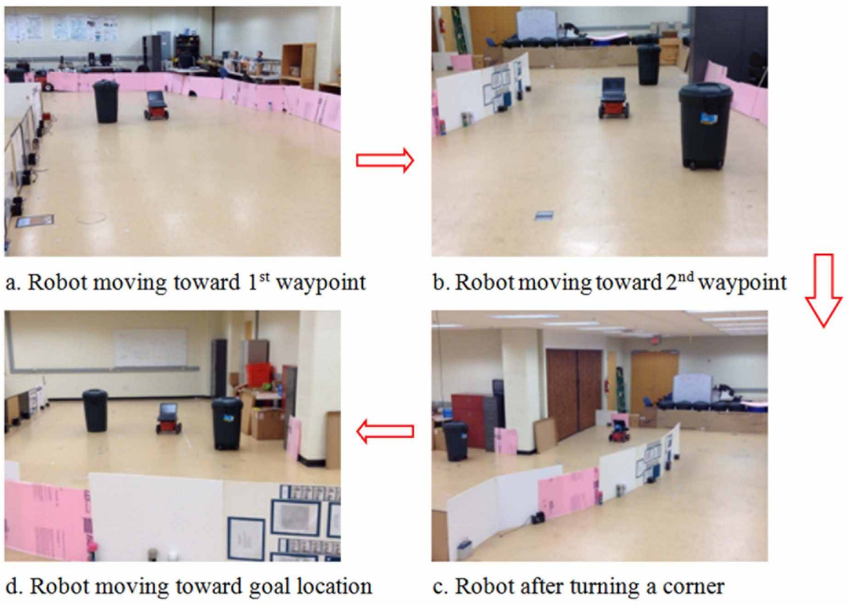
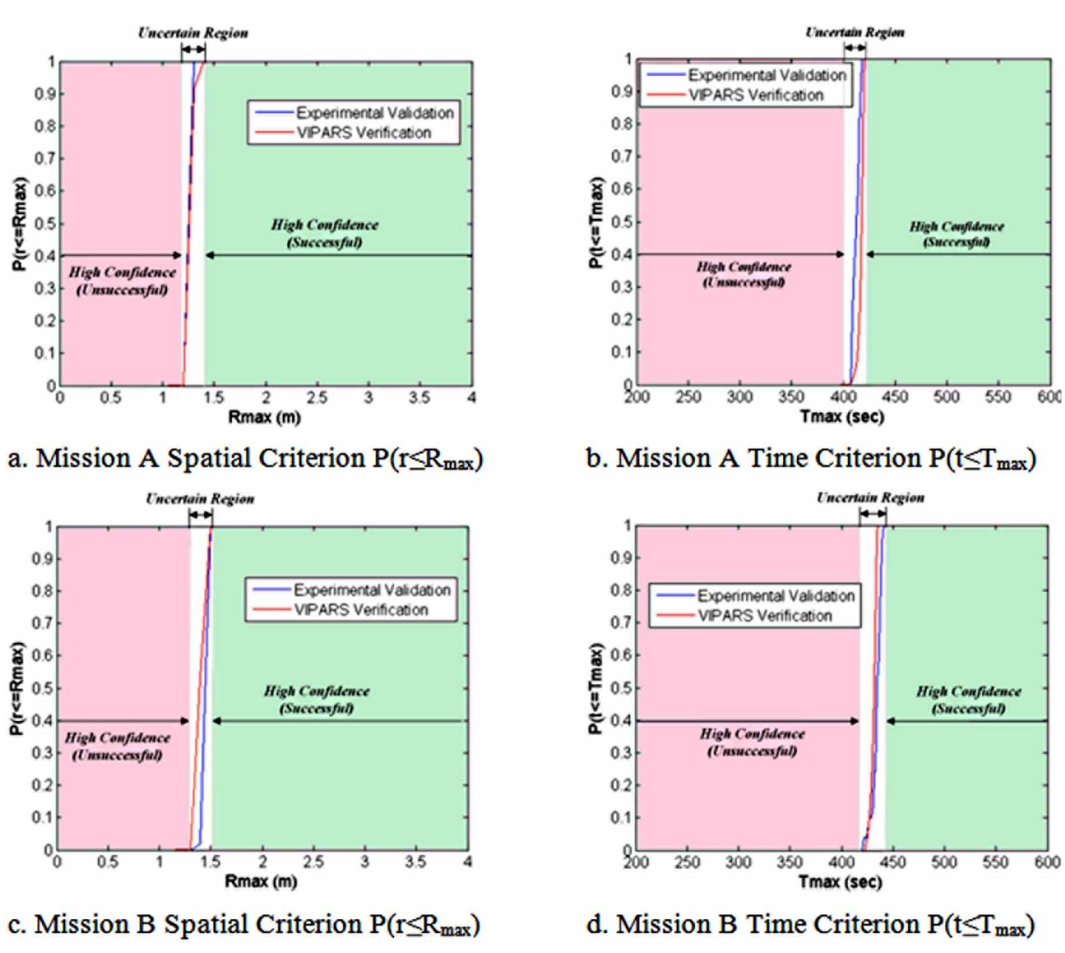

a. Robot moving toward 1st waypoint
b. Robot moving toward 2nd waypoint
c. Robot after turning a corner
d. Robot moving toward goal location

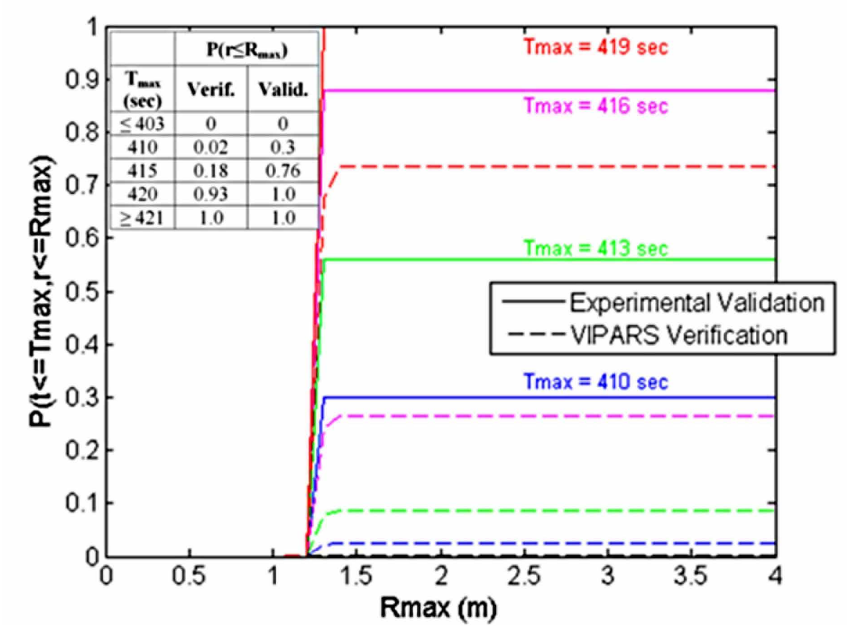
Figure 8. Results of VIPARS verification and experimental validation of spatial and time performance criteria for waypoint missions A and B



and the probability of mission success is between 0 and 1.0. As a result, the robot is not guaranteed to succeed with the mission. The High Confidence (Successful) is the region of near zero verification error and the mission is guaranteed to succeed with probability of 1.0. Consequently, the mission operator's decision for robot deployment can be based on which region the mission criteria fall into. For instance, if the specified performance criterion falls within the Unsuccessful region (e.g., R_{max} = 0.5m), the operator can either abort the mission or modify mission parameters or design.

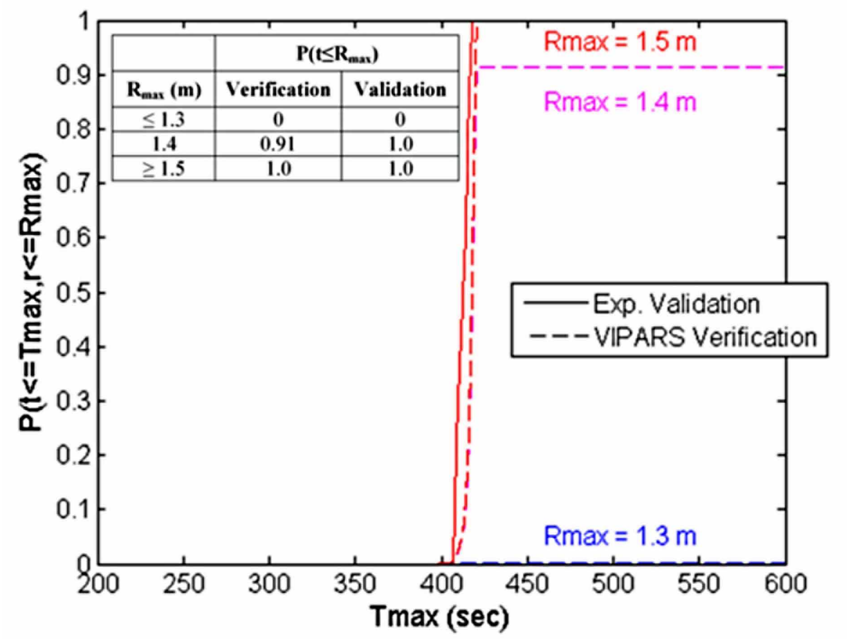
The overall mission success (Equation 5.1) is defined in terms of both spatial and time criteria. Thus, we examined further in Figures 9 and 10 the effects of various combinations of spatial and time criteria (R_{max} and T_{max}) on the mission success and verification error. The results can also be used to answer queries regarding the performance guarantee for a specific combination of T_{max} and R_{max}. Figure 9 shows the effects of the time criterion T_{max} on the V&V results of the spatial criterion $P(r \leq R_{max})$ for *Mission A*. While the T_{max}'s in both of its high confidence regions (Figure 8b) have no effect on the verification error for $P(r \leq R_{max})$, T_{max}'s that are in the *Uncertain* region (e.g., T_{max}= 415 sec) incur significant verification errors. For instance, for T_{max}=415sec, VIPARS predicted a success probability of 0.18, while the robot was actually successful 76% of the time in experimental trials.

Figure 9. V&V of spatial criterion at various Tmax for Mission A



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

Figure 10. V&V of time criterion at various Rmax for Mission A



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

Figure 10 shows the effects of the spatial criterion R_{max} on the V&V results of the time criterion $P(t \leq T_{max})$. While similar observations can be made here as in Figure 9, in this case, R_{max}'s have much less impact on the verification error of $P(t \leq T_{max})$ due to VIPARS's accuracy in predicting the spatial performance of mission even in the uncertain region (as shown in Figure 8a). Nonetheless, missions with performance criteria in the *Uncertain* regions should generally be avoided.

6. CONCLUSION

We have presented an approach to the problem of establishing a-priori performance guarantees for robot mission software for C-WMD robot missions in which failure could mean significant loss of life or severe property damage. In particular, we have addressed the issue of automatic verification of robot mission software that includes some preexisting software libraries that have to be included in the mission without being rewritten or annotated.

Two unique theoretical results were presented. The first theoretical results was the development of an environmental model representation for probabilistic maps, an indexed mixture of Gaussians, sufficient to support the reasoning necessary for automatic verification of missions that include probabilistic localization algorithms. The second theoretical result presented a collection of techniques for 'wrapping' existing software in a random variable envelope so that it could be included in probabilistic verification. A series of four wrapper functions – the mean-extended real function MER, the mean-extended normal function MEN, the extended normal function ENF, and extended real function ERF – were developed, each with different constraints and representational power.

Localization and mapping techniques intuitively offer advantages for robots navigating in unknown environments. This paper has applied *MissionLab*/VIPARS mission design and verification approach to the verification of autonomous behavior-based robot missions that use the ROS AMCL localization module. Verification result using the index mixture of Gaussian map representation and ERF wrapper function for ROS AMCL were presented for two different localization missions. The ERF wrapper was limited to a single member and a fixed variance was used. To completely implement the mixture extended function for the sampling-based approach in this paper, the full motion history for each sample request would need to be sent to the STDR node and AMCL reset between samples. The ability to cache these multiple sensory histories would improve computation time, but at the cost of directly instrumenting AMCL – a step we were avoiding for reasons discussed.

Experimental validation was also carried out for these two missions, and the verification and validation results compared to demonstrate the effectiveness of the approach. It is observed however, that VIPARS performs much better on the spatial performance criterion than on the time criterion. While the spatial random variable transformations are calculated using flow functions in a Bayesian network, time is represented only as discrete iterations of the Bayesian network. A key avenue of future investigation is to represent time as a random variable and have it also calculated using flow-functions, separating it from iteration so of the Bayesian network.

REFERENCES

Arkin, R. C. (1998). *Behavior-Based Robots*. Cambridge, MA: MIT Press.

Baeir, C., & Katoen, J.-P. (2008). *Introduction to Model Checking*. Cambridge, MA: MIT Press.

Bailey, T., & Durrant-Whyte, H. (June, September 2006). Simultaneous Localization and Mapping (DLAM): Party I, II. *IEEE Robotics and Automation Magazine*.

Blackman, S. (2004). Multiple Hypothesis Tracking for Multiple Target Tracking. *IEEE A&E Systems Magazine, 19*(1).

Brahman, J. (2009). *Verification and Analysis of Goal-Based Hybrid Control Systems* [Thesis]. P.D. California Institute of Technology, Pasadena, CA.

Cowley, A., & Taylor, C. (2011). Towards Language-Based Verification of Robot Behaviors. *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

Dellaert, F., Fox, D., Burgard, W., & Thrun, S. (1999). Monte Carlo localization for mobile robots. *Proceedings of the IEEE Int. Conf. on Rob. & Aut.,* Detroit.

DeMoura, L., & Bjorner, N. (2012). Satisfiability Modulo Theories: Introduction and applications. *Communications of the ACM*, *54*(9), 54–67.

Fisher, M., Dennis, L., & Webster, M. (2013). Verifying Autonomous Systems. *Communications of the ACM*, *56*(9), 84–93. doi:10.1145/2500468.2494558

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

Huang, J., Erdogan, C., Zhang, Y., Moore, B., Luo, Q., Sundaresan, A., & Rosu, G. (2014). ROSRV: Runtime Verification for Robots. *Proceedings of the 14th International Conference on Runtime Verification*, Toronto.

Jhala, R., & Majumdar, R. (2009). Software Model Checking. *ACM Computing Surveys*, *41*(4), 1–54. doi:10.1145/1592434.1592438

Jiang, S., & Arkin, R. (2015). SLAM-Based Spatial Memory for Behavior-Based Robots. *Proceedings of the 11th IFAC Symposium on Robot Control (SYROCO)*, Salvador, Brazil.

Katoen, J.-P. (2010). Advances in Probabilistic Model Checking. *Proceedings of the 11th International Conference VMCAI '10*, Madrid Spain.

Kiekbusch, L., Armbrust, C., & Berns, K. (2015). Formal Verification of Behavior Networks including Sensor Failures. *Robotics and Autonomous Systems*, *74*, 331–339. doi:10.1016/j.robot.2015.08.002

Kim, M., Kang, K.-C., & Lee, H. (2005). Formal Verification of Robot Movements - a Case Study on Home Service Robot SHR100. *Proceedings of the IEEE Int. Conf. Robotics and Automation*.

Klavins, E. (2004). A Language for Modeling and Programming Cooperative Control Systems. *Proceedings of the International Conference on Robotics and Automation*, New Orleans, LA. doi:10.1109/ROBOT.2004.1308780

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

Kwiatkowska, M., Norman, G., & Parker, D. (2004). Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal of Software Tools and Technology Transfer*, *6*(2), 128–142. doi:10.1007/s10009-004-0140-2

Leucker, M., & Schallhart, C. (2009). A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, *78*(5), 293–303. doi:10.1016/j.jlap.2008.08.004

Livingston, S., Murray, R., & Burdick, J. (2012). Backtracking temporal logic synthesis for uncertain environments. *Proceedings of the International Conference on Robotics and Automation*. doi:10.1109/ICRA.2012.6225208

Lyons, D., Arkin, R., Jiang, S., Harrington, D., & Liu, T. (2014). Verifying and Validating Multirobot Missions. *Proceedings of the IEEE/RSJ Int. Conf. on Robots and Systems*, Chicago.

Lyons, D., Arkin, R., Jiang, S., Harrington, D., Tang, F., & Tang, P. (2015). Probabilistic Verification of Multi-Robot Missions in Uncertain Environments. *Proceedings of the IEEE Int. Conf. on Tools with AI.,* Vietro sul Mare, Italy. doi:10.1109/ICTAI.2015.22

Lyons, D., Arkin, R., Jiang, S., Harrington, D., Tang, F., & Tang, P. (2016). Formal Performance Guarantees for Behavior-based Localization MIssions. *Proceedings of the IEEE Int. Conf. on Tools with AI*, San Jose CA. doi:10.1109/ICTAI.2016.0025

Lyons, D., Arkin, R., Jiang, S., Liu, T.-L., & Nirmal, P. (2015). Performance Verification for Behavior-based Robot Missions. *IEEE Transactions on Robotics*, *31*(3), 619–636. doi:10.1109/TRO.2015.2418592

Lyons, D., Arkin, R., Liu, T.-L., Jiang, S., & Nirmal, P. (2013). Verifying Performance for Autonomous Robot Missions with Uncertainty. *Proceedings of the IFAC Intelligent Vehicle Symposium*, Gold Coast, Australia. doi:10.3182/20130626-3-AU-2035.00034

Lyons, D., Arkin, R., Nirmal, P., Jiang, S., Liu, T.-L., & Deeb, J. (2013). Getting it Right the First time: Robot Mission Guarantees in the Presence of Uncertainty. *Proceedings of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Tokyo, Japan. doi:10.1109/IROS.2013.6697122

MacKenzie, D., Arkin, R., & Cameron, R. (1997). Multiagent Mission Specification and Execution. *Autonomous Robots*, *4*(1), 29–52. doi:10.1023/A:1008807102993

O'Brien, M., & Arkin, R. (2016). An Analysis of Displays for Probabilistic Robotic Mission Verification Results. *Proceedings of the 7th International Conference on Applied Human Factors and Ergonomics*, Las Vegas NV.

O'Brien, M., Arkin, R., Harrington, D., Lyons, D., & Jiang, S. (2014). Automatic Verification of Autonomous Robot Missions. *Proceedings of the 4th Int. Conf. on Simulation, Modelling and Prog. for Aut. Robots,* Bergamo, Italy.

Piterman, N., Pneuli, A., & Sa'ar, Y. (2006). Synthesis of Reactive(1) Designs. *Proceedings of the 7th International Conference VMCAI '06*, Charlestown SC.

Proetzsch, M., Berns, K., Schuele, T., & Schneider, K. (2007). Formal Verification Of Safety Behaviours Of The Outdoor Robot Ravon. *Proceedings of the 4th Int. Conf. on Informatics, Automation and Control*, Dortmund, Germany.

Ropertz, T., & Berns, R. (2014). Verification of behavior-based networks-using satisfiability modulo theories. *Proceedings of the 41st International Symposium on Robotics ISR/Robotik '14*.

Russel, S., & Norvig, P. (2010). *Artificial Intelligence*. Prentice-Hall.

Shankar, N. (2009). Automated Deduction for Verification. *ACM Computing Surveys*, *41*(4), 1–56. doi:10.1145/1592434.1592437

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic Robotics*. Cambridge, MA: MIT Press.

Trojanek, P., & Eder, K. (2014). Verification and testing of mobile robot navigation algorithms. *Proceedings of the IEEE/RSJ Int. Conf on Intelligent Robots and Systems (IROS),* Chicago.

Venet, A. (2008). A practical approach to formal software verification by static analysis. *ACM SIGAda Letters*, *27*(1), 92–95.

Walter, D., Taubig, H., & Luth, C. (2010). Experiences in Applying Formal Verification in Robotics. *Proceedings of the 29th International Conference on Computer Safety, Reliability and Security*, Vienna Austria. doi:10.1007/978-3-642-15651-9_26

Watkins, O., & Lygeros, J. (2003). Stochastic reachability for discrete time systems: an application to aircraft collision avoidance. *Proceedings of the IEEE Conf. on Decision and Control*, Maui, Hawaii. doi:10.1109/CDC.2003.1272482

Wongpiromsarn, T., & Murray, R. (2008). Formal Verification of an Autonomous Vehicle System. *Proceedings of the Conference on Decision and Control.*

Younes, H., & Simmons, R. (2002). Probabilistic verification of discrete event systems using acceptance sampling. *Proceedings of the 14th Int. Conf. on Computer Aided Verification*, Copenhagen Denmark. doi:10.1007/3-540-45657-0_17

Zaks, A., & Joshi, R. (2008). Verifying Multi-threaded C programs with SPIN. *Proceedings of the 15th International SPIN Workshop*, Los Angeles CA.

ENDNOTES

1 In process algebra, an interleaving theorem relates the sequential and parallel composition operations.

Damian M. Lyons is a Professor of Computer Science at Fordham University, and Director of Fordham's Robotics & Computer Vision Laboratory. He has degrees in Math, Engineering, and Computer Science from Trinity College, University of Dublin, Ireland, and a doctorate in Computer Science from the University of Massachusetts. His research interests include formal approaches to plan and program analysis and robot team exploration strategies. He was a senior researcher and department head at Philips Corporate Research, NY. He has served as Chair of the IEEE RAS TC on Assembly and Task Planning and is an IEEE Senior Member.

Ronald Arkin is Regents' Professor and Associate Dean in College of Computing at Georgia Tech. He served as STINT visiting Professor at KTH Stockholm, Sabbatical Chair at Sony IDL in Tokyo, and in the Robotics/AI Group at LAAS/CNRS in Toulouse. His research interests include behavior-based control and action-oriented perception, deliberative/reactive architectures, multiagent robotics, biorobotics, human-robot interaction, and robot ethics. He served on the Board of Governors of IEEE Society on Social Implications of Technology, IEEE Robotics and Automation AdCom, and founding co-chair of IEEE RAS TC on Robot Ethics. He is Distinguished Lecturer for the IEEE SSIT and IEEE Fellow.

Shu Jiang is a Robotics Ph.D. student in the Institute for Robotics and Intelligent Machines at Georgia Institute of Technology and a member of the Mobile Robot Laboratory. He received B.S. and M.S. degrees in Electrical Engineering from the University of Florida in 2009. His research interests include formal methods, human-robot team, educational and assistive robotics.

Matthew O'Brien is a Robotics Ph.D. student in the Institute for Robotics and Intelligent Machines at Georgia Institute of Technology and a member of the Mobile Robot Laboratory.

Feng Tang is a graduate student in the Robotics and Computer Vision Laboratory, Fordham University NY.

Peng Tang is a graduate student in the Robotics and Computer Vision Laboratory, Fordham University NY.