Establishing Performance Guarantees for Behavior-Based Robot Missions Using an SMT Solver

Feng Tang[1], Damian M. Lyons[1], Ronald C. Arkin[2]
[1]Fordham University, Bronx, NY, USA
[2]Georgia Institute of Technology, Atlanta, GA, USA

Abstract

In prior work [9] we developed an approach to formally representing behavior-based multi-robot programs, and the uncertain environments in which they operate, as process networks. We automatically extract a set of probabilistic equations governing program execution in that environment using a static analysis module called VIPARS, and solve these using a Dynamic Bayesian Network (DBN) to establish whether stated performance guarantees hold for the program in that environment. In this paper we address the challenge of expanding the range of performance guarantees that are possible by using an SMT-solver instead of a DBN.

We translate flow functions, which are recursive probabilistic expressions that capture program execution and that are generated by VIPARS, to formulas in the SMT-LIBv2 language. We use the *dReal* SMT-solver with a semi-decision procedure to check the satisfiability of the formulas. The experimental results show that our SMT approach succeeds in establishing performance guarantees for behavior-based robot missions at least as well as the DBN approach, but also for an example not possible with the DBN approach. Although the work presented here used VIPARS static analysis to generate flow functions, the approach we have taken could be used to leverage SMT for any similar probabilistic system.

1 Introduction

Autonomous multi-robot technology is playing a key role in Counter Weapons of Mass Destruction (C-WMD) and Urban Search and Rescue (USAR) missions since it has the potential to handle threats to human beings in a timely and safe fashion. Behavior-based robot programming is an important tool in autonomous robotics; it can yield programs that are robust to uncertainty about exactly what environment the robots will face during execution because it eschews any detailed modeling of the environment. This also means, however, that it may be difficult to predict exactly how the program will behave in any given environment. For this reason, automated analysis of behavior based programs for verification & validation (V&V) purposes has become a topic of research interest, e.g., [5] [16] [9]. Such program analysis can at best only produce an approximation of the performance due to the undecidability of the underlying problem of program verification, so a crucial issue is to understand what aspects of the problem to emphasize and what to simplify.

Lyons *et al.* [9] designed a probabilistic framework for verifying the performance of autonomous behavior-based robot missions in uncertain environments. The behavior-based mission is specified in *MissionLab* [12], a multi-agent robotics mission specification and control toolkit. It is translated from *MissionLab's* internal representation to a process-algebra notation PARS (Process Algebra for Robot Schemas). Environment models are also processes in this notation and Lyons *et al.* propose that a standardized set of environment models could be used to capture different classes of environment. The environment models in [9] address only uncertainty in motion and odometry, although obstacle uncertainty has been addressed for this approach in [10].

A behavior-based program and its environment is modeled by [9] as a set of interconnected, recurrent processes. A static analysis [14] module, VIPARS, is used to extract *flow functions* for each process - functions that capture how the process variables evolve as that process executes. An interleaving theorem is then introduced that allows the composition of all these individual flow functions, both for program and for environment, into a single *system flow function* capturing the interaction of program and environment variables. Process variables may be random variables and hence the system flow function may operate on distributions as well as values.

Establishing a performance guarantee for a robot mission in a given environment involves determining whether the system flow function with known initial conditions will ever lead to variable values (distributions) that satisfy the performance guarantee. Although they discuss more complicated performance guarantees, Lyons *et al.* [9] basically restrict their attention to the guarantee that a mission will achieve some criterion on environment variables (usually a spatial accuracy for a waypoint goal) with probability

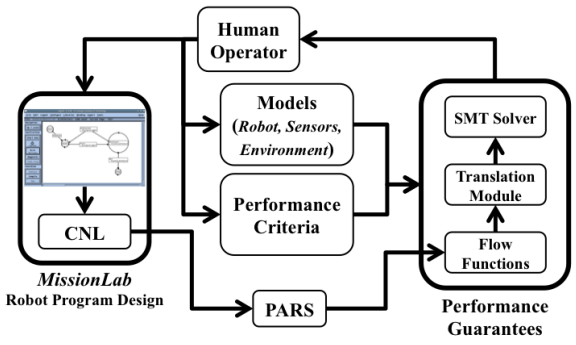

Figure 1 The framework of our approach. An SMT-solver is used to establish the performance guarantees for robot missions.

greater than a threshold before a time-limit has expired. They establish this using a Dynamic Bayesian Network (DBN))[17], constructed from the system flow function, as a probabilistic filter, with the time-limit as the upper bound on filter iterations. They demonstrate that this approach is fast and accurate when validated against physical executions. However, in this DBN approach all the initial conditions must be specified (even if as distributions) and it is limited to reasoning forwards from the initial conditions.

We have now developed a *Satisfiability Modulo Theories (SMT)* [4] solver approach to address the solution of the probabilistic system flow-function. The framework of the approach is shown in **Figure 1**. There are two advantages to leveraging SMT solvers in robotics problems of this kind. The first one is that it removes the constraint of fully specifying initial conditions and only reasoning forwards. For example, the initial position of the robot can be set in a specified area rather than a fixed position. The second one is that we are able to leverage state-of-the-art SMT solvers, e.g., *Z3* [6], *CVC4* [2], and *dReal* [8].

Applying SMT solvers to robotics research is not new. Witsch *et al.* [20] presented an incomplete SMT solver for non-linear continuous constraint satisfaction problems, and evaluated it and similar state-of-the-art solvers on problems originating in multi-robot system domains, e.g., task allocation or robot positioning. Saha *et al.* [18] proposed a compositional multi-robot motion planning framework that uses precomputed motion primitives for a group of robots and employs an SMT solver to synthesize trajectories for the individual robots. Nedunuri *et al.* [13] proposed an approach to solve integrated task and motion planning by transforming the problem into suitable input for an SMT solver, which can be used to symbolically explore the space of all integrated plans that correspond to paths in the placement graph, and also satisfy the constraints demanded by the plan outline and the requirements. Ropertz *et al.* [16] proposed a verification approach for complex behavior-based systems using SMT, which allowed for an automated system model modeling that enables an efficient and fine grained system analysis for ensuring safety and reliability.

The main contribution of our work here is threefold: (1) we take the advantage of SMT solvers to determine whether a performance guarantee holds automatically; (2) we use a semi-decision procedure which checks the satisfiability of recursive functions with lower bound and upper bound; (3) we extend the SMT solver to manipulate distributions.

2 Mission Analysis

Verification and validation of three typical robot missions is reported in [9][10][11]. The first mission is the *Back and Forth* mission, where the robot goes back and forth between positions A and B, and must reach each position within a certain time and spatial accuracy. It is difficult to succeed with this performance objective because uncertainties in robot motion result in the actual robot location drifting far from the desired location. The second one is the *Waypoint-based Biohazard Search* mission in which the task of the robot is to enter a building to search for a biohazard. The last one is the *Multi-robot Bounding Overwatch* mission, where the robots alternate in advancing forward and taking overwatch positions (i.e., covering for the advancing robot). PARS (Process Algebra for Robot Schemas) was introduced for representing these robot missions and as the underlying framework for reasoning about the performance guarantees for them. Software has since been developed in [15] to automatically translate robot missions from *MissionLab's* design environment into PARS.

2.1 PARS

PARS is a process algebra [1] for representing and analyzing robot programs and their interactions with an (uncertain) environment. Programs and environment models are specified in PARS as networks of communicating, concurrent processes. Introducing here only the most crucial PARS notation for space reasons, the robot controller process and environment process are written as *tail recursive* processes according to the straightforward process equation template:

$$\mathbf{P}\langle a\rangle = \mathbf{Q}\langle a\rangle\langle b\rangle ; \mathbf{P}\langle b\rangle . \qquad (1)$$

Eq. (1) defines a process $\mathbf{P}$ with input parameter a that repeats process $\mathbf{Q}$ until $\mathbf{Q}$ terminates in a 'fail state', an abort. The ";" is the sequential composition operator. Processes that evaluate conditions (e.g., whether the robot has reached a location) terminate in a success state if the condition is true and a fail state (abort) if the condition is false; a 'fail' in a sequential composition causes the composition to also 'fail'. In (1), process $\mathbf{Q}$ takes a as an initial parameter variable and produces b as a result variable. With every process $\mathbf{Q}$ we associate a *flow function* $f_{\mathbf{Q}}(a) = b$ that maps the input parameter variables of $\mathbf{Q}$ to its output variables. For *atomic* processes, f is predefined. For *composite processes*, processes defined as compositions of other processes, f has to be calculated by looking at the variable transformations and composition operations within the definition of $\mathbf{Q}$. The tail-recursive templates makes the calculation of flow functions much easier. In this way, a pro-

cess definition can be mapped to a composition of functional transformations on the variables of those processes. For example (1) can be mapped to a function composition $f_{\mathbf{P}} = f_{\mathbf{Q}}^n$, for $n \geq 0$.

In general, a process $\mathbf{Q}$ may have multiple variables, $v_1, v_2, \ldots, v_m \in V$, some of which can hold initial parameter values such as the starting location of the robot, its maximum velocity, etc., and some of which may deliver final result values such as position odometry or laser sensing (and some of which may be neither, and just used for intermediate values in calculations). We associate a flow function $f_{\mathbf{Q},v_i}$ with each variable in the process, and this function maps the product of the value sets of the variables in V to the value set of variable i. The flow-function for the process is then written somewhat more generally than we did in the last paragraph as:

$$f_{\mathbf{Q}}(V) = \left(f_{\mathbf{Q},v_1}(V), f_{\mathbf{Q},v_2}(V), \ldots, f_{\mathbf{Q},v_m}(V)\right). \quad (2)$$

The discussion here is a little simplified over the more complete description of PARS and flow-functions in [9] in that we do not discuss process termination, conditionals and loops and how to effect the calculation of the flow-function - we don't need this for the remaining discussion.

A variable in a process can be a random variable from a mixture of multivariate normal distributions $\boldsymbol{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), i = 1, ..., k$. For example, the initial position p_0 of the robot can be a two-dimensional random variable from a (single member mixture of) bivariate normal distribution(s) $\boldsymbol{N}\left(\boldsymbol{\mu}_{p_0}, \boldsymbol{\Sigma}_{p_0}\right)$.

2.2 VIPARS: Performance Analysis in PARS

A static analysis technique is developed in [9] to extract the system flow function from a system of interconnected parallel processes. The software that implements this technique is called VIPARS (Verification in PARS). The result of applying VIPARS to a system $\mathbf{S}$ of robot program and environment model is a set of functions f_v, one per variable in $\mathbf{S}$, that map values (distributions) in one iteration of the system to values (distributions) in the next. VIPARS leverages the fact that all the processes can be specified, without loss of generality, in a tail-recursive form as in Eq. (1).

Let us call $F(V)$ the function

$$F(V) = f_{\mathbf{S}}(V) = \left(f_{\mathbf{S},v_1}(V), f_{\mathbf{S},v_2}(V) \ldots, f_{\mathbf{S},v_m}(V)\right) \quad (3)$$

for $v_1, v_2, \ldots, v_m \in V$ being the (program and environment) variables in the system of program and environment $\mathbf{S}$. This probabilistic mapping can be represented as a Dynamic Bayesian Network (DBN)[17], and thus used to forward filter the values of the random variables. If V_0 is the initial values of the system variables, then $F(V_0)$ is the result of one application of this filter, and $F^n(V_0)$ is the result of n successive applications of the filter. Let $C(V)$ be a boolean function that tests whether the performance guarantee holds for the variables in V (e.g., whether the robot is in its goal location within a specific spatial accuracy). Then, establishing whether the performance guarantee holds just asks whether

$$\exists n < N_{max} \,.\, C(F^n(V_0)) \quad (4)$$

where N_{max} is the maximum allowed iterations of the filter. Each iteration of the filter models a fixed time step τ so if T is the time-limit for the mission, then $N_{max} = T_{max}/\tau$.

2.3 The Single-Waypoint Example

An example of such a tail-recursive system of processes and their analysis is the following from [9] and upon which the experiments in this paper are based. The robot controller process **MoveTo** takes the desired location to move to as an initial parameter g, and then recursively accepts a position input on its input port p and produces a velocity output $d(g - rp)$ proportional to the position error on its output port v until the error is smaller than a constant e:

$$\begin{aligned} \mathbf{MoveTo}\langle g\rangle \quad = \quad & \mathbf{In}\langle p\rangle\langle rp\rangle; \mathbf{Gtr}\langle |rp-g|, e\rangle; \\ & \mathbf{Out}\langle v, d(g-rp)\rangle; \mathbf{MoveTo}\langle g\rangle \end{aligned} \quad (5)$$

The environment the controller is carried out in here consists of the robot and its motion uncertainty. This can be represented as the tail-recursive process **Robot**:

$$\begin{aligned} \mathbf{Robot}\langle r, u\rangle \quad = \quad & (\mathbf{Delay}\langle\tau\rangle \,\#\, \mathbf{Out}\langle p, r * E_s\rangle \,\#\, \mathbf{At}\langle r\rangle); \\ & \mathbf{In}\langle v\rangle\langle u\rangle; \mathbf{Robot}\langle r + (u * E_m) * \tau, u\rangle \\ & E_m \sim \mathbf{N}(\mu_m, \Sigma_m),\ E_s \sim \mathbf{N}(\mu_s, \Sigma_s) \end{aligned} \quad (6)$$

The # operator is the PARS disabling composition, a parallel-min operator that terminates when the first of its arguments terminate, but otherwise behaves like a parallel composition. The two constants E_s, E_m are Gaussian distributions representing the odometry sensor noise and the velocity noise, and are measured by calibration experiments for the robot platform used in the experiments, a Pioneer 3-AT. The time constant τ is the fixed time step in this process. Each recursion of (6) advances the robot from r to $r + (u * E_m) * \tau$ where u is the velocity send to the robot. The robot can also report its current odometry reading as $r * E_s$.

The system composed of the parallel connected composition of these two processes generates the flow functions [9]:

$$\begin{aligned} f_{v1}(g) \quad &= \quad g \\ f_{v2}(r) \quad &= \quad r + (d(g - r * E_s) * E_m) * \tau \\ f_{v3}(u) \quad &= \quad d(g - r * E_s) \end{aligned} \quad (7)$$

Performance analysis consists of determining whether the robot position r will transform from an initial position p_0 to g with a specified spatial accuracy $a \leq |r - g|$ with probability $\leq P_T$ within time limit T_{max}, accomplished using a DBN $F(\{g, r, u\})$ built from Eq. (7).

But to solve this using an SMT-solver, rather than by DBN filtering, we need to first address the problem that the expressions in F can be operations on distributions, then translate F to the SMT-LIBv2 [3] language, and finally use an SMT solver to check the satisfiability of the formulas. We present a semi-decision procedure for checking the satisfiability of (4).

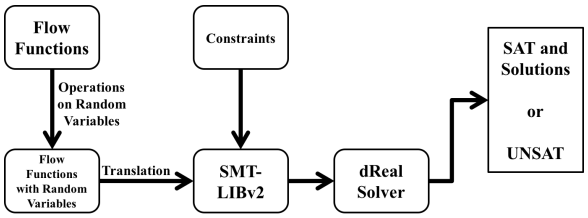

Figure 2 How the dReal solver is used to check the satisfiability of the formulas translated from flow functions.

3 Establishing Performance Guarantees Using SMT

We translate F and its operations on distributions to the SMT-LIBv2 language and operations on variables, and put constraints such as initial values, performance criteria and so forth on them. Then we will use an SMT-solver - *dReal* [8] to check the satisfiability of the formulas. This process is shown in **Figure 2**. We meet the challenge that $F^n(\cdot)$ has to be evaluated by presenting a semi-decision procedure.

3.1 Translating Operations on Distributions

We have chosen to only investigate random variables whose values distribute as represented by a mixture of multivariate normal distributions. Such variables play an important role in the specification of robot environments and in robot control programs, and are typically not well covered in the formal verification literature; that literature focuses more strongly on exponential distributions (for timing and queuing properties) and non-deterministic variables. For example, the initial position of the robot can be a two-dimensional random variable from a bivariate normal distribution. In that case, the interprocess communication ports, and variables – including initial variable values and result variables – must also be represented as random variables.

Flow functions that include random variables (e.g., eq. (7)) will include operations on those variables such as for example, the convolution of variables, or the scaling or offset of variables by values. However, the SMT solver we will use, *dReal*, does not support such operations on random variables, and therefore each operation must be translated to what *dReal* can handle.

Operations on distributions are translated to *operations on the parameters of those distributions*. A mixture is represented as a set of k normal distributions and weight values. Each normal distribution has a mean and variance parameter. Each operation on a random variable then translates to an operation on the mean and possibly an additional operation on the variance of each mixture member: a worst-case $2k$-fold increase in the number of operations. In general these are well known and straightforward mappings from distribution operations to operations on the distribution parameters. As examples of this step in translation:

- If F contains a convolution between two variables, this is translated to two operations: an addition of the means and an addition of the variances.
- If F includes the addition of a constant to a variable, it just translates to a single addition of the constant to the mean.

However, we need to regularize the set of operations so that the set of mixtures of normal distributions is closed with respect to the set of operations in F. This requires that some operations, such as the product of two random variables, be approximated rather than represented exactly. With this restriction, the syntactic translation step from distributions to distribution parameters can be easily automated. We will restrict ourselves to $k = 1$, singleton mixtures, in the remainder.

3.2 Translation to SMT Language

After obtaining flow functions and translating from operations on random variables to operations on distribution parameters, then the result needs to be rewritten into the SMT-LIBv2 Language, the input language for the SMT solver that we use, *dReal*, to check the satisfiability of (4). *dReal* is an SMT solver for first-order logic formulas over the reals. It can handle formulas with a wide range of nonlinear real functions such as polynomials, trigonometric functions, exponential functions and so on. SMT formulas over the real numbers are very hard to solve when nonlinear functions are involved. δ-complete decision procedures in *dReal* provided a new general framework for handling nonlinear SMT problems over the reals. More details about δ-complete decision procedures are in [7].

The main reason for using *dReal* is that even after translating to operations on distribution parameters, there is one other distribution-related complexity we need to handle: To evaluate the probability that the robot's location as given by it's position variable p is within a spatial accuracy of its goal location g, we need to evaluate the expression:

$$\int_{g-\varepsilon}^{g+\varepsilon} \boldsymbol{N}\left(x;\, \boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p\right) dx \tag{8}$$

where $\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p$ are the distribution parameters for the variable p and where ε represents a small area around the goal. This functionality needs non-linear arithmetic over the reals. **Code 1** is an example of calculating a probability density function using SMT-LIBv2.

Code 1 Calculating a 1-D probability density function.

```
(set-logic QF_NRA)
(set-info :precision 0.001)

(declare-fun x () Real)

(assert (<= 5.0 x))
(assert (<= x 15.0))
(assert (<= (/ (exp (/ (^ (- x 1) 2) (* 4 -2)))
(sqrt (* 4 (* 3.14159265 2)))) 0.3))

(check-sat)
(exit)
```

Algorithm 1 Semi-decision procedure for recursive functions.

Input: $f(\cdot)$, x_0, n, and g
Output: *SAT* and values of $x_k\,(k=0,1,\ldots,i)$, or *UNSAT*

Initialisation :
1: $i=1$
2: **while** $i \leq n$ **do**
3: $formula = (x_1 == f(x_0)) \wedge \ldots \wedge (x_i == f(x_{i-1})) \wedge (x_i == g)$
4: check the satisfiability of $formula$ using an SMT-solver
5: **if** *SAT* **then**
6: **return** *SAT* and the values of $x_k\,(k=0,1,\ldots,i)$
7: **end if**
8: $i = i+1$
9: **end while**
10: **return** *UNSAT*

3.3 Semi-Decision Procedure

All system flow functions are generated from tail-recursive processes and describe the relationship between the process variables values from one recursion to the next. For example, the flow function $f_{\mathbf{Q}}(p)$ describes the relation between the value of the variable p in process $\mathbf{Q}$ at time k, p_k and that at time $k+1$, p_{k+1}: $p_{k+1} = f_{\mathbf{Q}}(p_k)$. In this example, let us assume p_k is the position of the robot at any time k. Hence the relation between the initial position p_0 at time 0 and the position p_n at time n can be written as $p_n = f_p^n(p_0)$. Our aim is to leverage an SMT solver to determine if there exist values of the process variables that satisfy the assigned constraints, but state-of-the-art SMT solvers don't support recursive functions well. Inspired by *Leon* [19], we developed an efficient semi-decision procedure for checking the satisfiability of these system flow functions with certain constraints. *Leon* is an automated system for verifying, repairing, and synthesizing functional Scala programs. The core solving engine of *Leon* is a semi-decision procedure for first-order constraints with recursive functions.

The procedure is shown in **Algorithm 1**. Given a recursive function $x_{i+1} = f(x_i), (i = 0,1,\ldots)$, the initial values of x_0, upper bound n, and the value of goal g, check whether there exists a k that satisfies the specified constraints $x_k == g$ and $k \leq n$ using an SMT-solver.

3.4 Translating the Single-Waypoint Mission

In the results section, we will be looking at results for a single-waypoint mission in which a robot moves from its initial position P_0 to a goal location G. We select a probability threshold criterion P_T for the robot arriving at the goal position G successfully (evaluated in SMT using (8)), and we establish the maximum time that the robot can use to reach G. We also set the minimum speed of the robot for the mission and the calibrated uncertainty parameters that quantify robot motion.

The robot mission software used for this mission is a behavior-based waypoint program constructed in *MissionLab*, which ultimately translate into processes and a system flow function similar to that in section 2.3. All of the a-priori information is expressed in the form of SMT formulas along with the translated system flow function. We then use *dReal* to check the satisfiability of the mission.

One advantage of the SMT-based approach over the Dynamic Bayesian Network based approach for determining the probability of mission success is that some initial values don't need to be fixed. For example, the initial value of the robot position P_0 can be assigned in a range rather than fixed values. **Code 2** is a simplified example of a mission where a robot moves one step in one dimensional space without the fixed initial values of the variables.

Code 2 A simple example of a mission.

```
; declare variables
(declare-fun pos_0_mean () Real)
(declare-fun pos_0_var () Real)
(declare-fun pos_1_mean () Real)
(declare-fun pos_1_var () Real)
(declare-fun speed () Real)
(declare-fun prob () Real)

; constraints on the variables
(assert (>= pos_0_mean 0.0))
(assert (<= pos_0_mean 100.0))
(assert (>= pos_0_var 0.0))
(assert (<= pos_0_var 500.0))
(assert (= speed 50.0))
(assert (>= prob 0.8))

; code for calculating the
; new position of the robot
(assert (= pos_1_mean (...) ))
(assert (= pos_1_var (...) ))

; code for calculating the probability that
; the robot arrives at the goal position
(assert (= prob (...) ))

(check-sat)
(exit)
```

4 Results

Our empirical results include two kinds of single-waypoint missions. In the first, the initial position of the robot, P_0, is fixed, and we compare the preliminary results with the Dynamic Bayesian Network (DBN) approach, showing that the SMT approach works as well as the DBN approach. In the second, we only give a range of the initial position P_0, and check whether the robot can arrive the goal position G. In all, the performance criterion is that the robot has a cumulative probability of $\boldsymbol{P}_T = 80\%$ or greater of reaching the goal position G before a maximum time T_{max}. We select $N_{max} = T_{max}/\tau = 10$.

Our results show the SMT approach can provide performance guarantees beyond the scope of the DBN method, but with a significant time penalty.

4.1 Single-Waypoint Mission with Fixed Initial Position

In the first single-waypoint mission, the nominal initial position of the robot is fixed as

$$\boldsymbol{\mu}_{P_0} = (5500mm, 5000mm), \tag{9}$$

and the goal position is

$$G = (6500mm, 5000mm), \tag{10}$$

which means the robot moves $1000mm$ from its initial location toward G. We use bivariate normal distribution to represent the initial position P_0 as

$$\boldsymbol{N}\left(\boldsymbol{\mu} = (5500.00, 5000.00), \boldsymbol{\Sigma} = \begin{bmatrix} 500.00 & 0 \\ 0 & 500.00 \end{bmatrix}\right). \tag{11}$$

The variance values here indicate the accuracy with which the initial position is known. So while we say the initial position is fixed, in fact it is known only with this specified accuracy.

Using the proposed approach, the SMT engine indicates that the performance criterion is satisfied for $n = 8 < N_{max}$ and the final position distribution is

$$\boldsymbol{N}\left(\boldsymbol{\mu} = (6450.36, 5001.45), \boldsymbol{\Sigma} = \begin{bmatrix} 7402.84 & 0 \\ 0 & 565.82 \end{bmatrix}\right). \tag{12}$$

The increase in variance in the final position is due to the motion uncertainty model, indicating a potential wide spread of final locations for the robot. The uncertainty is much larger in the direction of motion (along the X axis). For a much fuller discussion of the modeling of uncertainty and interpretation of mission verification results, see [9].

Table 1 shows the mean of the positions of the robot in each time step predicted by our SMT-based and DBN-based approach. Both approaches generate very similar results, validating the SMT implementation. The execution time of the DBN approach is 1.5 seconds and that of the SMT approach is 10 seconds. For the SMT approach, there are 64 variables and 65 formulas in the satisfied case. The second mission will show that why we are studying on the SMT approach although it is much slower than the DBN approach.

4.2 Single-Waypoint without Fixed Initial Position

In the second single-waypoint mission, the initial position of the robot is not fixed, its mean is $\boldsymbol{\mu}_{P_0} = (\mu_x, \mu_y)$, where $5250 \leq \mu_x \leq 5750$ and $4750 \leq \mu_y \leq 5250$. The goal position is still $G = (6500mm, 5000mm)$. Specifying a range for an initial variable value means a forward filtering approach is no longer possible.

The output of the SMT approach is that the performance guarantee is satisfied for the mean of the start position at $(5681.28, 4750.22)$ and the mean of the final position at $(6367.81, 4963.45)$ for $n = 6 < N_{max}$. This kind of analysis can not be carried out with the DBN approach, since that demands the initial position be a distribution and be a prespecified input value. Here it is loosely constrained initially and its value is actually an output of the verification. This is a useful tool for a mission designer and therefore a useful addition to verification. The execution time of this mission is 180 seconds due to the additional computation incurred with the looser constraint on the robot initial position.

Table 1 The Positions of the Robot Generated by SMT and DBN Approach for the First Single-Waypoint Mission.

Time Step	SMT	DBN
0	(5500.00, 5000.00)	(5500.00, 5000.00)
1	(5676.91, 5001.33)	(5676.91, 5001.33)
2	(5832.96, 5002.25)	(5832.96, 5002.26)
3	(5970.61, 5002.82)	(5970.61, 5002.83)
4	(6092.01, 5003.08)	(6092.02, 5003.10)
5	(6199.10, 5003.06)	(6199.12, 5003.09)
6	(6293.56, 5002.79)	(6293.58, 5002.83)
7	(6376.87, 5002.28)	(6376.90, 5002.32)
8	(6450.36, 5001.45)	(6450.39, 5001.49)

Table 2 The Time Complexity Table for the Second Single-Waypoint Mission.

Time Step	#Variables	#Formulas	Time	Result
0	8	9	0.007	UNSAT
1	16	17	0.052	UNSAT
2	24	25	0.271	UNSAT
3	32	33	1.265	UNSAT
4	40	41	1.831	UNSAT
5	48	49	3.097	UNSAT
6	56	57	6.602	UNSAT
7	64	65	7.116	UNSAT
8	72	73	24.761	UNSAT
9	80	81	87.341	UNSAT
10	88	89	192.454	UNSAT
11	96	97	1934.088	UNSAT
12	104	105	80.573	SAT

4.3 Time Complexity

We tested the time complexity of SMT for a single waypoint mission to check the relation between time complexity and the number of variables and formulas for the mission. **Table 2** shows the relation among time step n, number of formulas, number of variables, execution time and result for the second single-waypoint mission. The time complexity of time step is exponential. Note that when the result is *SAT*, which means the robot arrives the final position with a certain probability, the time cost drops down.

5 Conclusion

In prior work we addressed the problem of verifying performance guarantees for a behavior-based robot carrying out a mission in an uncertain environment. In our ap-

proach, a mission designer used the *MissionLab* robot mission design environment to construct robot mission software. The mission designer can choose to verify whether the mission when executed in a modeled uncertain environment, would achieve a performance guarantee. The modeled information contains information on the motion uncertain of the robot, and the performance guarantee typically asks whether the robot can reach its destination with a specific spatial accuracy and within a time limit. To implement this, we developed an approach to formally representing behavior-based multi-robot programs and uncertain environments as process networks, automatically extracting a set of probabilistic equations to establish performance guarantees for the program using the VIPARS module, and solving these using a Dynamic Bayesian Network (DBN) [9].

In this paper, we used an SMT-solver in place of the DBN to expand the range of performance guarantees possible in our prior work. We used VIPARS to extract the system flow function for a single-waypoint robot mission. Then all these flow functions were translated into the SMT-LIBv2 language with certain constraints. Finally we used the *dReal* solver and a semi-decision procedure to check the satisfiability of these recursive formulas. The SMT approach can provide performance guarantees beyond the scope of the DBN approach; however, the SMT approach runs much slower than the DBN approach for the same missions. Although the work presented here used VIPARS static analysis to generate flow functions, the approach we have taken could be used to leverage SMT for any similar probabilistic system.

Future work will focus on: (1) Improving the efficiency of solving recursive functions using an SMT-solver so as to establish performance guarantees for more complicated behavior-based robot missions such as multi-robot missions. (2) Translating from flow functions into SMT-LIBv2 Language automatically. (3) Adding Gaussian distribution and Gaussian mixture model as theories to SMT. (4) Introducing new performance criteria for behavior-based robot missions according to the range of performance guarantees expanded by our SMT-solver approach. For example, a robot spending minimum time to arrive at the goal position without fixing its initial position.

Acknowledgment

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

6 References

[1] J. C. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2):131–146, 2005.

[2] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. Cvc4. In *Computer Aided Verification*, pages 171–177. Springer, 2011.

[3] C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0. Technical report, Department of Computer Science, The University of Iowa, 2010. Available at `www.SMT-LIB.org`.

[4] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. *Handbook of Satisfiability*, 185:825–885, 2009.

[5] A. Cowley and C. Taylor. Towards language-based verification of robot behaviors. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011.

[6] L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[7] S. Gao, J. Avigad, and E. M. Clarke. δ-complete decision procedures for satisfiability over the reals. In *Automated Reasoning*, pages 286–300. Springer, 2012.

[8] S. Gao, S. Kong, and E. M. Clarke. dreal: An smt solver for nonlinear theories over the reals. In *Automated Deduction–CADE-24*, pages 208–214. Springer, 2013.

[9] D. M. Lyons, R. Arkin, S. Jiang, T.-M. Liu, and P. Nirmal. Performance verification for behavior-based robot missions. *Robotics, IEEE Transactions on*, 31(3), 2015.

[10] D. M. Lyons, R. Arkin, T.-M. Liu, S. Jiang, and P. Nirmal. Verifying performance for autonomous robot missions with uncertainty. In *8th IFAC Symposium on Intelligent Autonomous Vehicles*, 2013.

[11] D. M. Lyons, R. C. Arkin, P. Nirmal, and S. Jiang. Designing autonomous robot missions with performance guarantees. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2583–2590. IEEE, 2012.

[12] D. C. MacKenzie, R. C. Arkin, and J. M. Cameron. Multiagent mission specification and execution. In *Robot Colonies*, pages 29–52. Springer, 1997.

[13] S. Nedunuri, S. Prabhu, M. Moll, S. Chaudhuri, and L. E. Kavraki. Smt-based synthesis of integrated task and motion plans from plan outlines. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 655–662. IEEE, 2014.

[14] F. Nielsom, H. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 2010.

[15] M. O'Brien, R. C. Arkin, D. Harrington, D. M. Lyons, and S. Jiang. Automatic verification of autonomous robot missions. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 462–473. Springer, 2014.

[16] T. Ropertz and K. Berns. Verification of behavior-based networks - using satisfiability modulo theories. In *ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of*, pages 1–6. VDE, 2014.

[17] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2009.

[18] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia. Automated composition of motion primitives for multi-robot systems from safe ltl specifications. In *Intelligent Robots and Systems (IROS), 2014 IEEE/RSJ International Conference on*, pages 1525–1532. IEEE, 2014.

[19] P. Suter, A. S. Köksal, and V. Kuncak. Satisfiability modulo recursive programs. In *Static Analysis*, pages 298–315. Springer, 2011.

[20] A. Witsch, H. Skubch, S. Niemczyk, and K. Geihs. Using incomplete satisfiability modulo theories to determine robotic tasks. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 4784–4789. IEEE, 2013.