Submitted, 2009 Hybrid Systems: Computation and Control (HSCC)
http://www.cds.caltech.edu/~murray/papers/2008s_hm09-hscc.html

Probabilistic Safety Analysis of Sensor-Driven Hybrid Automata

Julia M. B. Braman and Richard M. Murray

Dept. of Mechanical Engineering, California Institute of Technology
braman@caltech.edu

Abstract. The control programs of complex autonomous systems that have conditional branching can be modeled as linear hybrid systems. When the state knowledge is perfect, linear hybrid systems with state-based transition conditions can be verified against a specified unsafe set using existing model checking software. This paper introduces a formal method for calculating the failure probability due to state estimation uncertainty of these sensor-driven hybrid systems. Problem complexity is described and some reduction techniques for the failure probability calculation are given. An example goal-based control program is given and the failure probability for that system is calculated.

1 Introduction

Autonomous robotic missions generally have complex, fault tolerant control systems. There are several ways to incorporate the necessary fault tolerance in a control architecture. One way is to create a flexible control system that can reconfigure itself based on the state of the system and environment in the presence of faults. However, if the control system cannot be analyzed for safety in the presence of estimator error, the added complexity of the system's reconfigurability could reduce the system's effective fault tolerance.

One particularly useful way to model a fault tolerant control system is as a hybrid system. When the continuous dynamics of these systems are sufficiently simple, it is possible to verify that the execution of the hybrid control system will not fall into an unsafe regime [1]. There are several software packages that can be used for this analysis, including HyTech [2], UPPAAL [3], and PHAVer [4], all of which are symbolic model checkers. PHAVer in particular is able to exactly verify linear hybrid systems with piecewise constant bounds on continuous state derivatives. Safety verification for fault tolerant hybrid control systems ensures that the occurrence of certain faults will not cause the system to reach an unsafe state.

However, these verification software packages cannot handle uncertainty in the estimated state variables involved in mode transition logic. For autonomous systems, none of the state variables used in the control system are known perfectly, and these uncertainties can affect the safety of the system. Stochastic hybrid systems include uncertainty in the transitions of the hybrid automaton

as probabilistic transition conditions. Many papers have been written on the verification of stochastic hybrid systems. For example, Prajna et al [5] use barrier certificates to bound the upper limit of the probability of failure of the stochastic hybrid system and Kwiatkowska et al [6] discuss a probabilistic symbolic model checking software called PRISM. However, purely probabilistic transition conditions do not model estimation uncertainty in the constrained state variables well; deterministic transitions with probabilistic components may be a better model.

In this paper, a formal method for determining the failure probability due to estimation uncertainty of a verifiable hybrid system with sensor-driven, state-based transitions, which builds on the work in [7], is given. These particular hybrid systems are derived from goal-based control programs called goal networks. Mission Data System (MDS), a goal-based control architecture developed at the Jet Propulsion Laboratory, is the basis for the design of these goal networks [8]. The goal networks are then converted using a bisimulation to linear hybrid automata and verified against an unsafe set using existing symbolic model checking software such as PHAVer [9]. The discrete mode transitions in the hybrid automata are then based on the state variables of the system estimated from sensor values.

As an illustration of the type of tasks analyzed, consider the sensor-based planning problem shown in Fig. 1. This rover, which must safely follow the path to the specified end point, has three main sensor systems: GPS, LADAR, and an inertial measurement unit (IMU). The speed limit along the path is dependent on the combined health of these sensors. Each sensor degrades or fails in a specified way. The GPS can experience periods of reduced accuracy (e.g. satellite dropouts) or failure (electrical or structural signal interference), and these can both be modeled as recoverable stochastic events. The LADAR health depends on the location of the sun in the sky and whether the robot is in dusty conditions; however it will also be modeled as degrading and recovering stochastically. Finally, the IMU health is dependent on the temperature of the device. If the temperature of the IMU is too low, a heater can be used to heat the sensor. If the IMU temperature gets too high, the unit must be powered off. For the safety of the system, it is important to verify the goal networks for this problem against the following unsafe set:

1. The rover is not stopped when the IMU is off and the GPS is degraded.
2. The rover moves forward when the LADAR unit is compromised.

The methods described in this paper allow the analysis of the safety of the control program against the given unsafe set when the estimation of the sensor health values is not perfect. If these states were known exactly, a traditional hybrid system verification would be a sufficient test of the safety of this system. However, a full analysis of this system must include the calculation of the failure probability due to estimation uncertainty, for which a method is introduced here.

The paper is organized as follows. A summary of goal network specifications and the conversion procedure can be found in Section 2. Section 3 is the main

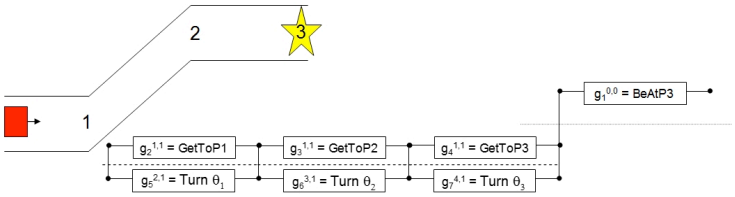

Fig. 1. A depiction of the example task where the rover must traverse a path and reach the end point, which is marked with a star, and the goal network associated with it.

contribution of this paper, the failure probability calculation for the converted linear hybrid systems, which formalizes and extends the process developed in [7]. Section 4 describes the complexity of this calculation and lists some reduction techniques. Section 5 is the failure probability calculation for the example introduced above, and Section 6 summarizes the contributions and describes directions for future work.

2 Background Information

2.1 Linear Hybrid Automata

There are several symbolic model checkers available that are capable of verifying linear hybrid automata. A linear hybrid automaton H consists of the following components [2]:

1. A finite, ordered list of continuous state variables, $X = \{x_1, x_2, ..., x_n\}$, and their time derivatives, $\dot{X} = \{\dot{x_1}, \dot{x_2}, ..., \dot{x_n}\}$.
2. A control graph, (V, E), where V is the set of control modes or locations of the system, and E is the set of control edges or transitions between the different modes of the system.
3. The set of invariants for each location, $inv(v)$, the set of initial conditions for each location, $init(v)$, and the set of flow conditions, $dif(v, \dot{X})$, where $v \in V$.
4. The set of transition labels, Σ, and transition actions or reset equations, A.

These components describe a linear hybrid system that can be successfully verified using HyTech or PHAVer. The reachability analysis used in the safety verification of these hybrid automata finds the set of all states that are connected to a given initial state by a valid run. This can cause a huge explosion of the state space, however, so symbolic model checkers partition the state space into sets that are similar in the given reachability analysis. For example, given some interesting condition, PHAVer will return the set of the state space in which it is reachable; these interesting conditions given to the software are generally "unsafe" conditions for the particular system.

2.2 Goal Network Conversion and Verification Procedure

A software control architecture developed at the Jet Propulsion Laboratory uses state models and reconfigurable goal-based control programs for the control of autonomous systems [10]. Using MDS, systems are controlled by networks of goals, which directly express intent as constraints on physical states over time. Unlike the more traditional sequences of commands used to control robotic space missions, goal networks allow for branching in the execution plan so that more fault tolerance is built into the control program at the cost of added complexity [11]. In the goal network, there are parent goals that elaborate different tactics, or methods, of accomplishing the parent goal's state constraint. Whether a tactic is elaborated in a goal network execution depends on the states of certain state variables that are passively constrained in the tactics. A passive constraint on a state variable simply means that goal success depends on the present state of the state variable; active, or controlled, constraints on state variables are goals in which control action is employed for the success of the goal.

Let $\mathcal{G}$ be the set of all goals in a goal network, where $\mathcal{G} = G \cup U$. The set $G = \{g_1^{i_1,j_1}, g_2^{i_2,j_2}, ..., g_N^{i_N,j_N}\}$ consists of all controlled goals in a goal network, where i_n is the parent goal index and j_n is the tactic number into which the goal is elaborated, for $n = 1, ..., N$. The set of passive goals is $U = \{u_1^{i_1,j_1}, u_2^{i_2,j_2}, ..., u_M^{i_M,j_M}\}$, where i_m is the index of the goal's parent goal (which is always a controlled goal) and j_m is the goal's tactic number for $m = 1, ...M$. Let $\mathcal{T} = \{T_1, T_2, ..., T_{K+1}\}$ be the set of time points in the goal network, where $T_1 < T_2 < ... < T_{K+1}$ for increasing time. Each goal $g_n^{i_n,j_n}$ and $u_m^{i_m,j_m}$ has several associated functions:

1. start() returns the goal's starting time point.
2. end() returns the goal's ending time point.
3. svc() returns the state variable constrained by the goal.
4. cons() returns the constraint (or invariant) on the state variable; cons() $\in Q \times \mathbb{R}$, where $\mathbb{R}$ is the set of real numbers and $Q = \{=, \neq, <, >, \leq, \geq\}$.

The time points of the goal network are constrained to be well-ordered which means that the time points fire in strictly increasing order. Due to the structure imposed on the time points, goals can be grouped into K groups, G_k, $k = 1, ..., K$ where

$$G_k = \{g_n^{i_n,j_n} \in G | \text{start}(g_n^{i_n,j_n}) \leq T_k \wedge \text{end}(g_n^{i_n,j_n}) \geq T_{k+1}\}. \quad (1)$$

A similar definition exists for U_k.

A valid execution of the goal network consists of a sequence of alternating flow and transition conditions,

$$\phi_{I_f}(t_f)...\phi_{I_2}(t_2)\rho_{I_2 I_1}\phi_{I_1}(t_1)X_0, \quad (2)$$

where X_0 is the set of initial conditions of the controlled state variables, $\phi_{I_n}(t_n)$ is the set of flow conditions associated with the executable set of goals θ_{I_n} and propagated forward in time t_n steps, $\rho_{I_{n+1}I_n}$ is the transition between the executable sets of goals θ_{I_n} and $\theta_{I_{n+1}}$, and I_n is the set of indices of the goals associated with a certain executable set. Executable sets are subsets of the goals

that can be executed at the same time. The set of all executable sets that are built from the goals in G_k is Θ_k. Completion goals are controlled goals $g_n^{i_n,j_n}$ with non-trivial exit conditions; each executable set $\theta_I \in \Theta_k$ either contains all completion goals in the group or has exit conditions based on goal failure only.

There are two different types of transitions between the goals in the goal network, transitions based on goal completion, $\rho_{IJ,k}^c \in S^{comp}$, and transitions based on goal failure, $\rho_{IJ,k}^f \in S^{fail}$, where S^{comp} and S^{fail} are the sets of all completion and failure transitions, respectively. The transition $\rho_{IJ,k}^c$ is from $\theta_I \in \Theta_k$ to $\theta_J \in \Theta_{k+1}$, and its conditions are based on the completion goals in the group. The transition $\rho_{IJ,k}^f$ is from $\theta_I \in \Theta_k$ to $\theta_J \in \Theta_k$ or to Safing (in which case, the transition is labeled $\rho_{ISafe,k}^f$). For both $\rho_{IJ,k}^c$ and $\rho_{IJ,k}^f$, only transition conditions that evaluate to true are taken.

The conversion procedure between goal networks and linear hybrid automata is a bisimulation and is more fully described in [9]. First, the hybrid automaton created from the goal network contains all possible executions of the goal network in its paths. The locations v_I are created directly from each of the executable sets θ_I, and transitions $\tau_{J,k}^s$, $\tau_{I,k}^e$, and $\tau_{IJ,k}^f$ are created directly from $\rho_{IJ,k}^c$ and $\rho_{IJ,k}^f$. Next, other hybrid automata are created from the models of the state variables that are constrained by passive goals. This linear hybrid automata is then analyzed using an existing symbolic model checking software against an unsafe set to verify that the goal network never reaches the unsafe state. A goal network whose linear hybrid automata passes the test is then called verified.

3 Failure Probability Calculation

3.1 Automata Specification and Models

The hybrid automaton created from the goal network, called the 'goals' automaton, has execution paths of the form

$$\psi_{I_f}(t_f)\tau_{I_f I_{f-1}}...\psi_{I_2}(t_2)\tau_{I_2 I_1}\psi_{I_1}(t_1)X_0 \tag{3}$$

where X_0 is the set of initial conditions on the controlled state variables, $\psi_{I_n}(t_n) = dif(v_{I_n}, X)(t_n)$ is the flow associated with location v_{I_n} for t_n time steps, and $\tau_{I_n I_{n-1}}$ is the transition from location v_{I_n} to $v_{I_{n-1}}$. The locations of the hybrid automaton have the same group structure as do the goals in the goal network, with disjoint sets of locations, $V_1, V_2, ..., V_K$ where $\bigcup_{k=1}^K V_k = V$ and where $V_k \equiv \Theta_k$.

Each location $v_I \in V_k$ directly corresponds with an executable set $\theta_I \in \Theta_k$; therefore, the location v_I is associated with a set I_g of controlled goals and a set I_u of passive goals. The dynamics of a location are derived from the constraints of the controlled goals, which include completion goals. Each location v_I has a function, pcons(), associated with it, where pcons(v_I) returns the set C_I of passive goal constraints associated with that location, $C_I = \{\text{cons}(u_m^{i_m,j_m}) | u_m^{i_m,j_m} \in \theta_I\}$. The goals automaton has three types of transitions associated with it: entry transitions, $\tau_{J,k}^s$, are transitions from the preceding group (or initially for $k = 1$);

exit transition, $\tau^e_{I,k}$, are transitions from location v_I in group V_k to locations in group V_{k+1} (or to the Success location for $k = K$); failure transitions, $\tau^f_{IJ,k}$, are transitions from location v_I to location v_J, where $v_I, v_J \in V_k$. Some failure transitions may instead go from location $v_I \in V_k$ to Safing; in that case, the transition is labeled $\tau^f_{ISafe,k}$. These failure transitions are the sensor-driven state-based transitions conditions for which the failure probability is calculated.

Failure probability can be calculated for a subset of all verifiable goal networks. An additional restriction on the hybrid automaton is that all failure and entry transition conditions must be entirely state-based; failure and entry conditions cannot be based on the order of tactics attempted. This restriction is not a serious one; in general, state-based transitions are a characteristic of more robust control programs. Due to the construction of the hybrid automaton [9], these entry and failure transitions for location v_I are based on the passive goal constraints associated with that location. The uncertain state variables are the set of state variables $\mathcal{U}_k$ such that $\mathcal{U}_k = \{svc(u_m^{i_m,j_m}) | \forall u_m^{i_m,j_m} \in U_k\}$.

It is assumed that certain statistical information is known about the system. For each of the uncertain state variables in set $\mathcal{U}_k$, there must be a way to model the propagation of the state variable as a stationary Markov process; since these state variables are not controlled, oftentimes this approximation is a good one. Also, for each uncertain state variable, there must be some measure of how good its state estimator is. With this information, the failure probability of the goal network with respect to the unsafe set may be calculated.

3.2 Failure Path Specification

The calculation of failure probability means nothing unless the conditions of goal network failure are specified; these failure conditions are called the unsafe set.

Definition 1. *An* unsafe set *is a set of failure conditions* $Z = \{z_1, z_2, ..., z_n\}$ *where each* z_j *has associated functions:*

1. $plc(z_j, \gamma_i)$ *returns the set of unsafe location constraints,* α*, on state variable* $\gamma_i \in \Gamma$ *where* $\Gamma = \{svc(u_m^{i_m,j_m}) | \forall u_m^{i_m,j_m} \in U\}$ *and where* $\alpha \subset \Lambda_i$*, the set of all possible locations (or discrete sets of states) for the passively constrained state variable* γ_i*.*
2. $loc(z_j, v_I)$ *returns true if the unsafe condition applies in location* v_I *and false if it does not.*

Definition 2. *A* complete system state *is both the estimated and actual states,* $\lambda_{ij,e}$ *and* $\lambda_{ij,a}$ *of each uncertain state variable* $\gamma_i \in \Gamma$ *at a point in time in a possible execution of the goal network. The set of all possible complete system states is* $\mathcal{S}$*. Each complete system state has two functions associated with it.*

1. *est*() *returns the estimated states* $\lambda_{ij,e}$ *of each uncertain state variable* $\gamma_i \in \Gamma$*.*
2. *act*() *returns the actual states* $\lambda_{ij,a}$ *of each uncertain state variable* $\gamma_i \in \Gamma$*.*

Definition 3. *A system state s (actual or estimated)* satisfies *a set of state constraints C_s if*

$$s \wedge C_s \neq \mathit{False}. \tag{4}$$

A function $sat(s, C_s)$ will return true if s satisfies C_s and false otherwise.

Lemma 1. *Let $\Omega_k \subset \mathcal{S}$ be the set of complete system states that drive the automata execution from group V_k into the unsafe set Z. For a complete system state $\omega \in \mathcal{S}$, $\omega \in \Omega_k$ if and only if there exists $z_j \in Z$ and $v_I \in V_k$ such that*

$$sat(act(\omega), plc(z_j)) \wedge sat(est(\omega), pcons(v_I)) \wedge loc(z_j, v_I) \tag{5}$$

is true.

Proof. Let $\omega \in \Omega_k$ but assume there is no v_I that satisfies (5). By the definition of the unsafe set, there must exist some $z_j \in Z$ such that $\mathrm{sat}(\mathrm{act}(\omega), \mathrm{plc}(z_j))$ is true, since entrance into the unsafe set is always driven by the actual system state. Since the unsafe set specifies the total system state, including the location in the goals automaton, there must exist some v_I such that $\mathrm{loc}(z_j, v_I)$ is true. To enter location v_I and thus the unsafe set from complete system state ω, $\mathrm{sat}(\mathrm{est}(\omega), \mathrm{pcons}(v_I))$ must be true since the transitions in the automaton are state driven by definition and because the estimated state drives the transitions in the hybrid automata execution. Therefore, v_I does satisfy (5). The other direction of the proof is obvious by the definition of the unsafe set. □

Since the hybrid automaton has been verified against the unsafe set in the perfect knowledge case, the only way for the execution of the goal network to reach the unsafe set is by a difference between the actual and estimated state of the system. The remaining complete system states fall into two sets. If the estimated system state causes the automaton to transition into the Safing location, the complete system state is $f \in F_k$, where F_k is the set of all safing system states. Finally, all other complete system states, ξ, belong to the set of nominal system states, Ξ_k. Figure 2 shows the graphical breakdown of system state sets.

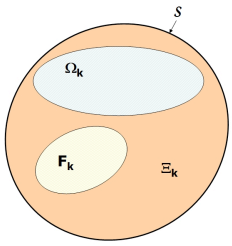

Fig. 2. A representation of sets of complete system states.

Definition 4. *A* failure path *in group V_k is a sequence of complete system states that has the form $\xi_1\xi_2...\xi_{n-1}\omega_n$, where $n = 1, ..., r$ and the value of r depends on the completion time and type of the group.*

The number and length of the failure paths in each group depends on several characteristics of the group which will be described in the following sections.

3.3 Completion Time

The completion time of a group V_k depends on the completion goals in the group. The type of completion time (uniform or non-uniform) depends on the presence of rate goals that affect the completion goals in the group.

Definition 5. *A* nominal execution path *of a group V_k is a path $\xi_1\xi_2...\xi_r \in N_k$ in which only nominal complete system states are visited before the group is exited and execution continues in group V_{k+1}. The set of all nominal execution paths in group V_k is N_k.*

Definition 6. *Given the set of nominal execution paths N_k for group V_k, the* completion time*, c_k, is the minimum length of nominal execution path,*

$$c_k = \min_{\nu \in N_k} length(\nu). \tag{6}$$

The completion time of a group is the time it takes to achieve the group's completion goals at the fastest constrained rate. If there are no completion goals in a group and instead there is a time constraint imposed on the bounding time points, the completion time is simply the time constraint imposed.

Definition 7. *In a* uniform completion *group, V_k,*

$$c_k = \min_{\nu \in N_k} length(\nu) = \max_{\nu \in N_k} length(\nu). \tag{7}$$

The uniform completion case holds in groups that either have only one rate goal or have no rate goals and a time constraint. If there is only one rate goal in a group, all locations will contribute the same amount towards the achievement of the completion goals. Likewise with a time constraint, each time step in any location in the group contributes the same amount towards the completion of the time constraint. Another way to define the uniform completion time case is that the contribution values of each location in the group are the same.

Definition 8. *The* contribution value *of a location $v_I \in V_k$, $val(v_I) \in \mathbb{R}$, is the normalized contribution towards the achievement of the completion goals in V_k that location v_I gives each time step. In uniform completion groups, for each $v_I \in V_k$, $val(v_I) = 1$.*

Definition 9. *A* non-uniform completion *group is one in which*

$$\min_{\nu \in N_k} length(\nu) \neq \max_{\nu \in N_k} length(\nu). \tag{8}$$

In the non-uniform completion case, for each location $v_I \in V_k$, $\text{val}(v_I) \leq 1$. In this case, the group would have more than one rate goal that contributes to the achievement of the completion goals.

3.4 Probability Calculations

The failure probability is calculated from the sum of the probabilities of all the failure paths in a group; the paths in a group depend on the completion time in that group, and the procedure for finding all the failure paths in a group depends on whether it is a uniform or non-uniform completion group. The failure paths for the uniform completion case are easy to find; the procedure for the non-uniform completion case is more difficult. Both will be described in this section. First, let $\text{loc}(\xi, k) \in V_k$ be a function that returns the location associated with the nominal complete system state ξ in group V_k.

Lemma 2. *For every failure path $\pi = \xi_1\xi_2...\xi_{r-1}\omega_r \in \Pi_k$, where Π_k is the set of all failure paths in group V_k,*

$$\sum_{i=1}^{r-1} val(loc(\xi_i, k)) < c_k. \tag{9}$$

Proof. The proof of this lemma is simple; if the sum of the contribution values of the nominal states visited in a failure path equals or exceeds c_k, the execution continues into the next group by the definition of completion time. □

For both uniform and non-uniform completion groups, the failure probability of the group, $W_s(k)$ is the sum of the path probabilities of all the failure paths in that group,

$$W_s(k) = \sum_{\Pi_k} P(\pi). \tag{10}$$

For the uniform completion case, collections of stationary and transition probabilities between nominal and unsafe system states can be created. These probabilities are calculated from the stationary Markov processes that model the state propagation of each uncertain state variable and the estimation uncertainty model for each state variable that only depends on the actual state of the state variable. First, let the probability of executing a failure path of length one in group V_k be a_k, where

$$a_k = \sum_{\Omega_k} P(\omega). \tag{11}$$

Let W_k be a vector of probabilities whose elements are the stationary probabilities of each nominal system state, $\xi_i \in \Xi_k$, where $W_k(i) = P(\xi_i)$. Let Q_k be the matrix of transition probabilities between all nominal system states. Let $W_{u,k}$ be the vector of transition probabilities from each nominal system state to all the unsafe system states,

$$W_{u,k}(i) = \sum_{\Omega_k} P(\omega_r|\xi_{i,r-1}). \tag{12}$$

Proposition 1. *The failure probability for the uniform completion case in group V_k can be calculated using the following formula, for $c_k \in [2, \infty)$,*

$$W_s(k) = a_k + W_k \cdot (\sum_{i=0}^{c_k-2} Q^i) W_{u,k}. \tag{13}$$

When $c_k \to \infty$, the equation becomes

$$W_s(k) = a_k + W_k \cdot (I - Q)^{-1} W_{u,k}. \tag{14}$$

Proof. The failure probability is the sum of all the failure path probabilities; for the uniform completion case and the definitions of a_k, W_k, Q_k, and $W_{u,k}$ given above, equation (13) sums the path probabilities of all failure paths of length one to length c_k. If a path has length c_k+1, c_k of the path elements must be nominal states; because for the uniform completion case, $\text{val}(\xi) = 1$ for all $\xi \in \Xi_k$ and Lemma 2, a path of length c_k+1 is not possible in group V_k. Therefore, equation (13) is the sum of all possible failure paths in V_k. Using

$$\sum_{i=1}^{\infty} Q^i = (I - Q)^{-1}, \tag{15}$$

one can derive (14) from (13). □

The infinite completion time case is equivalent to finding the stationary probability of the expanded system state Markov chain. It is essentially one minus the probability of entering the Safing location from group V_k.

For non-uniform completion groups, path length depends on the order in which the locations are visited. However, the problem is able to be simplified by collecting locations into sets based on their contribution values. Then, failure paths can be found based on these sets of locations, and vectors and matrices of stationary and transition probabilities for each set of locations can be defined like the ones defined above for the uniform completion case.

Let $B = \{b | b = \text{val}(v_I), \text{ for all } v_I \in V_k\}$, where all $b \in B$ are unique (for all $b_i, b_j \in B, b_i \neq b_j$) and ordered ($B = \{b_1, b_2, ..., b_m\}$ such that $b_1 > b_2 > ... > b_m$). Since $\text{val}(v_I)$ is the rate of location $v_I \in V_k$ normalized by the maximum rate in group V_k, $b_1 = 1$. Now, let $\beta_i = \{\xi_j | \text{loc}(\xi_j, k) = v_I \wedge \text{val}(v_I) = b_i, \forall \xi_j \in \Xi_k\}$. Then, failure paths can be created using β_i instead of using the individual nominal system states, where for failure path $\beta_{i_1}\beta_{i_2}...\beta_{i_{r-1}}\omega_r$,

$$\sum_{j=1}^{r-1} b_{i_j} < c_k. \tag{16}$$

Like in the uniform completion case, the path probabilities can be summed over the unsafe system states and the probabilities of the nominal system states in each contribution value set can be collected into vectors and matrices.

Since the failure probability of a group is always the sum of the failure path probabilities, the only real difference between the two types of groups is how to

find the failure paths. In the uniform completion group, all nominal system states could be placed in set β_1; so, all failure paths could be made from zero up to $c_k - 1$ instances of β_1 followed by some $\omega \in \Omega_k$. For the non-uniform completion case, the order of the β sets in the failure path matters. The following basic procedure finds all of the failure paths of a non-uniform completion group.

1. Add all paths of the form $\beta_i\omega$ to the set Π_k of failure paths in group V_k.
2. For all β_i where $i = 1, ..., m$ and m is the number of sets β, and for all failure paths $\pi_j \in \Pi_k$, add β_i to the beginning of path π_j.
   (a) If path $\beta_i\pi_j = \pi_l$ for some $\pi_l \in \Pi_k$, discard the new path.
   (b) If $b_i + \sum_{\beta_l \in \pi_j} b_l \geq c_k$, discard the new path.
   (c) Else, add the new path to Π_k.
3. Repeat step 2 until no new failure paths are added to Π_k.

This procedure could be more efficient, but it is guaranteed to find all possible failure paths.

The overall hybrid automata failure probability can be calculated by summing the probabilities of all the failure paths through the automata. To do this, the probability of each group reaching the Safing location, $W_f(k)$, must be calculated using the same procedure as described above with states f replacing states ω at the end of the "failure" paths. The probability of traversing group V_k nominally is then

$$W_n(k) = 1 - W_s(k) - W_f(k). \tag{17}$$

Proposition 2. *The failure probability of the system of $K > 1$ groups is given by*

$$W_s = W_s(1) + \sum_{i=2}^{K} (\prod_{j=1}^{i-1} W_n(j)) W_s(i). \tag{18}$$

Proof. The failure probability of the hybrid system is the sum of the failure path probabilities through the system. Since the failure paths can only consist of zero to $K - 1$ nominal group transitions followed by a failure, (18) gives all failure paths through the hybrid system. Any entrance into Safing removes the execution from the hybrid system and precludes failure in the future, and so is excluded from the failure probability calculation. □

4 Problem Complexity and Reduction Techniques

The quality of the failure probability calculated above depends only on the quality of the probability models used; the procedure is exact in that it finds all failure paths and calculates the probability of each. The problem of finding all the path probabilities increases in complexity exponentially fast. Because the paths can be collected into similar sets in both the uniform and non-uniform completion cases, the complexity explosion is mostly due to the number of uncertain state variables and the number of states of each of those state variables.

Let $y(\gamma_i)$ be the number of states in Λ_i for each uncertain state variable $\gamma_i \in \Gamma$. Then, the number of complete system states goes as

$$\prod_{\Gamma} y(\gamma_i)^2. \tag{19}$$

In the non-uniform completion case, the number of distinct path sets depends on the number of rate goals in the group and the completion time c_k; however, this generally increases less quickly than the number of complete system states for a system under control.

There are two ways to reduce the number of complete system states for a system, and therefore decrease the complexity of the failure probability calculation. The first is to form derived state variables from the existing uncertain state variables. A derived state variable is a non-physical uncertain state variable whose state propagation completely depends on one or more other uncertain state variables. An example of this would be a system health state variable that is based on all of the sensor health state variables for the system. If there is a way to replace all instances of several uncertain state variables in the goals automaton with a derived state variable that is based on those uncertain state variables, the derived state variable can be used in the failure probability calculation instead. An example of this technique is described in the next section.

The second way to reduce the complexity of the failure probability calculation is to leverage the state models of the uncertain state variables by taking advantage of dependent state variables. A dependent state variable is an uncertain state variable that has model dependencies on the state of at least one other state variable. In some cases, only certain complete system states are possible because of these model dependencies. That will produce zero probability of being in these impossible system states; these system states can then be removed from the overall set $\mathcal{S}$ of complete system states. An example of this complexity reduction technique can also be found in the next section.

5 Rover Example

To find the failure probability of the verified goal network described in the introduction and in [9], it is noted that the calculation can be done for group V_1 only due to the similarity between the first three groups and the inability of reaching the unsafe set in the fourth. The goal network as verified has four uncertain state variables (IMU Temperature, GPS Health, LADAR Health, and IMU Health), each with three possible state values. Since that translates into $3^8 = 6561$ complete system states, simplification is necessary. Instead, if the System Health state variable with three states replaces the sensor health state variables, and a two state IMU Health is used, the number of complete system states reduces to $2^2 * 3^4 = 324$.

Another observation is that the IMU Health depends on the IMU Temperature. Since there is a model that controls what the IMU Health is estimated to be given the IMU Temperature, the estimated IMU Health is known given

the IMU Temperature. However, the actual IMU Health may not always be known given the actual IMU Temperature due to modeling errors. In certain cases, such as when the estimated IMU Temperature causes the IMU Power to be turned OFF, that the actual IMU state is known given the estimated IMU Temperature. This dependence of an uncertain state variable on another causes the number of elements to be further reduced. Dependencies between two state variables is dealt with by creating a new "state variable" that consists of all the possible actual and estimated states that the two variables can take given the dependencies. For this problem, the new state variable is called TI and has 18 states, which are made up of estimated and actual values of IMU Temperature and IMU Health. With the nine possible states for the estimated and actual values of the System Health state variable, the new total number of complete system states is $18 * 9 = 162$.

The group V_1 is a non-uniform completion group with three different contribution values, 1, 1/2, and 0, that correspond to the three speed limit goals. The nominal set, Ξ_1, has 120 complete system states, the Safing set, F_1, is empty, and the unsafe set, Ω_1 contains the remaining 42 elements. With a completion time $c_k = 5$, a set of probabilities for the propagation of the TI and System Health state variables, and given estimation uncertainties for the TI state variable, the estimation uncertainty of the System Health state was varied and the failure probability for group V_1 was calculated. The results are shown in Fig. 3.

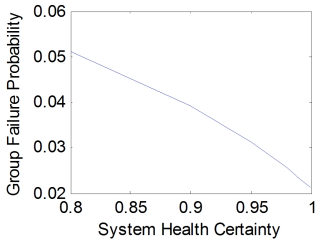

Fig. 3. Group failure probability vs. SH estimation uncertainty

6 Conclusions and Future Work

A formal method for calculating the probability of a verifiable sensor-driven hybrid system entering into a specified unsafe set due to estimation uncertainty was presented. The hybrid system is derived from a goal network control program for an autonomous system; the calculation of the failure probability of this system gives the designer some information about the original goal network. If the failure probability of a given system is too high for the design requirements, several changes could be made. First, the estimator for the state variable could be improved; for some cases, a better sensor could be used to reduce the probability of failure; and finally, the goal network could be designed to depend less on a relatively unknown state variable.

Future work on this problem includes extending the reduction techniques outlined in Section 4 expand the use of this calculation to more complex systems. Also, there may be ways to design the original system so that the failure probability due to estimation uncertainty is reduced if the quality of the sensors and estimators is known in advance. The verification of goal networks in the presence of different forms of uncertainty, including estimation uncertainty, is an important problem, and this approach seems promising as a design tool for goal-based control programs.

Acknowledgements

The authors would like to gratefully acknowledge Michel Ingham, David Wagner, Robert Rasmussen, and the MDS team at JPL for feedback, and MDS instruction. This work was funded by NSF and AFOSR.

References

[1] R. Alur, T. Henzinger, and P.-H. Ho, "Automatic symbolic verification of embedded systems," *IEEE Transactions on Software Engineering*, vol. 22, no. 3, pp. 181–201, 1996.

[2] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "HyTech: A model checker for hybrid systems," *International Journal on Software Tools for Technology Transfer*, 1997.

[3] K. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1-2, pp. 134–152, 1997.

[4] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past HyTech," *International Conference on Hybrid Systems: Computation and Control*, 2005.

[5] S. Prajna, A. Jadbabaie, and G. J. Pappas, "Stochastic safety verification using barrier certificates," *IEEE Conference on Decision and Control*, 2004.

[6] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with PRISM: a hybrid approach," *Int J Software Tools Technology Transfer*, vol. 6, pp. 128–142, 2004.

[7] J. M. Braman and R. M. Murray, "Safety verification of fault tolerant goal-based control programs with estimation uncertainty," *American Control Conference*, 2008.

[8] M. Ingham, R. Rasmussen, M. Bennett, and A. Moncada, "Engineering complex embedded systems with State Analysis and the Mission Data System," *AIAA Journal of Aerospace Computing, Information and Communication*, vol. 2, pp. 507–536, December 2005.

[9] J. M. Braman and R. M. Murray, "Automatic conversion software for the safety verification of goal-based control programs." Submitted, International Conference on Software Engineering, 2009.

[10] D. Dvorak, R. Rasmussen, G. Reeves, and A. Sacks, "Software architecture themes in JPLs Mission Data System," *IEEE Aerospace Conference*, 2000.

[11] R. D. Rasmussen, "Goal-based fault tolerance for space systems using the Mission Data System," *IEEE Aerospace Conference Proceedings*, vol. 5, pp. 2401–2410, March 2001.