Safety Verification of a Fault Tolerant Reconfigurable Autonomous Goal-Based Robotic Control System

Julia M. B. Braman, Richard M. Murray, and David A. Wagner

Abstract— **Fault tolerance and safety verification of control systems are essential for the success of autonomous robotic systems. A control architecture called Mission Data System (MDS), developed at the Jet Propulsion Laboratory, takes a goal-based control approach. In this paper, a method for converting goal network control programs into linear hybrid systems is developed. The linear hybrid system can then be verified for safety in the presence of failures using existing symbolic model checkers. An example task is simulated in MDS and successfully verified using HyTech, a symbolic model checking software for linear hybrid systems.**

I. Introduction

Autonomous robotic missions by nature have complex control systems. In general, the necessary fault detection, isolation and recovery software for these systems is cumbersome and added on as failure cases are encountered in simulation. There is a need for a systematic way to incorporate fault tolerance in autonomous robotic control systems. One way to accomplish this could be to create a flexible control system that can reconfigure itself in the presence of faults. However, if the control system cannot be verified for safety, the added complexity of the reconfigurability of a system could reduce the system's effective fault tolerance.

Mission Data System (MDS) is a software control architecture that was developed at the Jet Propulsion Laboratory [1]. It is based on a systems engineering concept called State Analysis [2]. Systems that use MDS are controlled by goals, which directly express intent as constraints on physical states over time. By encoding the intent of the robot's actions, MDS has naturally allowed more fault response options to be autonomously explored by the control system [3].

A great deal of work to date has focused on detecting and recovering from sensor failures in the control of autonomous systems [4]. Several fault tolerant control architectures for autonomous systems have been developed in which the control effort is layered to deal with faults on different levels, including low levels of hardware control and high levels of supervisory control [5], [6]. Fault diagnosis can be handled by modeling complex systems as stochastic hybrid systems with modes that account for failure states. The failures can then be detected using multiple-model based hybrid estimation schemes [7] or by using variations of traditional particle filters to aid in the accurate estimation of low probability but high risk failure modes [8]. Although many fault tolerant control systems achieve reconfigurability, few actually change the commands given to the system. One system uses adaptive neural/fuzzy control to reconfigure the control system in the presence of detected faults [9], and another reconfigures both the control system design and the inputs to the control system [10], although neither adjusts the intent of the commands in response to failures.

One particularly useful way to model fault tolerant control systems is as hybrid systems. Much work has been done on the control of hybrid systems [11]. 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 [12]. There are several software packages available that can be used for this analysis, including HyTech [13], UPPAAL [14], and VERITI [15], all of which are symbolic model checkers. HyTech in particular is used for checking linear hybrid automata, where the dynamics of the continuous variables can be modeled by linear differential inequalities that take the general form of $\mathbf{A}\dot{\mathbf{x}} \leq \mathbf{b}$ [12]. 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.

In this paper, MDS is used as a goal-based control architecture for a representative robotic task involving sensor failures and goal re-elaboration. The major contribution of this paper is the extension of a process to convert complex goal networks with several state variables and goal elaborations into hybrid automata that can be verified for safety using existing symbolic model checking software. An example goal network is developed in MDS, converted to a hybrid automaton, and then verified in the presence of sensor failures. A more detailed description of this work is available as a technical report [16].

The structure of this paper is as follows. Section II summarizes important concepts of MDS which pertain to this work. Section III introduces the example task, system design, and goal network. Section IV describes the major contribution of this work, the general process for converting goal networks into hybrid automata. Section V returns to the example, discussing simulation results as well as the hybrid automata that were created and the results of the safety verification. Finally, Section VI concludes the paper and discusses future directions of research.

II. Mission Data System Overview

A. State Analysis

State Analysis is a systems engineering methodology that focuses on a state-based approach to the design of a system

J. Braman and R. Murray are with the Dept. of Mech. Eng., California Institute of Technology, Pasadena, CA 91125, USA braman@caltech.edu

D. Wagner is a Senior Software Engineer with the Flight Software Applications Group at the Jet Propulsion Laboratory, Pasadena, CA, USA

[2]. In State Analysis, the control system and the system under control are considered separately. Models of state variable effects in the system under control are used for such things as the estimation of state variables, control of the system, planning, and goal scheduling. State variables are representations of states or properties of the system that are to be controlled or that affect a controlled state. Examples of state variables could include the position of a robot, the temperature of the environment, the health of a sensor, or the position of a switch.

Using State Analysis, the state variables of the system under control are identified. A model of the system under control is developed and controllers and estimators are designed using the models. Goals and goal elaborations are created, also based on the models. Goals are specific statements of intent used to control a system by constraining a state variable in time. Goals are elaborated from a parent goal based on the intent and type of goal, the state models, and several intuitive rules, as described in [2].

B. Mission Data System

A core concept of State Analysis is that the language used to design the control system should be nearly the same as the language used to implement the control system. Therefore, the software architecture, MDS, is closely related to the systems engineering theory described above.

Data structures called software state variables are central to MDS [17]. A state variable can contain much information; for example, a position state variable for a robot in the plane could contain the robot's (x,y) position, its velocity in component form, and uncertainty values for each piece of information. Each state variable has a unique estimator, and if necessary, a controller. Goals can be created that constrain some or all of a state variable's information. For example, a goal could constrain the velocity of the position state variable used in the previous example, but could leave the position or uncertainties unconstrained.

Goal networks replace command sequences as the control input to the system. Goal networks consist of a set of goals with their associated starting and ending time points and temporal constraints. A goal may cause other constraints to be elaborated on the same state variable and/or on other causally related state variables. These goals must have an associated elaboration class. The elaboration class instructs the elaborator in MDS to add certain goals to the goal network in support of the parent goal. The goals in the goal network and their elaborations are scheduled by the scheduler software component so that there are no conflicts in time, goal order or intent. The scheduled goals are then achieved by the estimator or controller of the state variable that is constrained.

Elaboration allows MDS to handle tasks more flexibly than control architectures based on command sequences. One example is fault tolerance. Re-elaboration of failed goals is an option if there are physical redundancies in the system, many ways to accomplish the same task, or degraded modes of operation that are acceptable for a task. The elaboration class for a goal can include several pre-defined tactics. These tactics are simply different ways to accomplish the intent of the goal, and tactics may be logically chosen by the elaborator based on programmer-defined conditions. This capability allows for many common types and combinations of faults to be accommodated automatically by the control system [3].

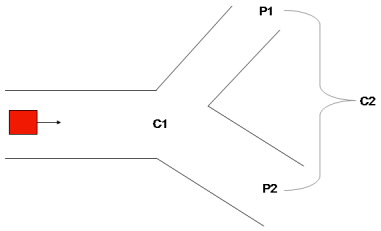

Fig. 1. Simulated robotic task

III. Example Task Design

This section describes the design of an autonomous robotic system, task, and a goal network that will accomplish the task in the presence of sensor failures. This example illustrates some of the MDS principles outlined in the previous section.

A. Task Design

An autonomous robotic task is considered in which a simulated robot with several sensors follows a path within a given uncertainty bound. The task could be compared to a Mars scientific mission that has two points of interest (p_1 and p_2); the first is more desirable but needs a lower uncertainty in the robot's position to reach it. The mission is considered a success if the robot does not wander off the path (where it could be damaged or get stuck), and the mission is completed if the robot reaches either point of interest. As shown in Figure 1, the planned route for the simulation consists of two checkpoints, c_1 and c_2; after the first checkpoint, c_1, there are two possibilities for the location of c_2, p_1 and p_2. The first of these possibilities, p_1, lies down a path that has a somewhat tighter error bound and requires a higher standard of sensor health. The other possibility, p_2, lies down a second path that allows for a larger error bound and a somewhat degraded sensor capability.

The path is successfully navigated by the robot if the robot stays within the path boundaries, representing the error bounds allowed down each path. Completion of the task occurs when the robot navigates to and stops sufficiently near c_2 without breaching the boundary. The second checkpoint, c_2, is first assigned to be at location p_1, but can be changed to be p_2 upon the failure or degradation of critical sensors.

B. System Design

The robot used in this simulation is equipped with three sensors: differential GPS, LADAR, and odometry (the collection of position, orientation, and velocity information deduced from wheel encoders). These three sensors are

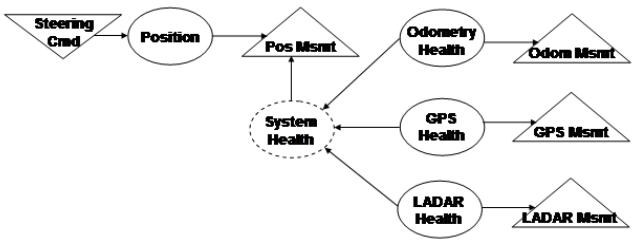

Fig. 2. State effects diagram; solid ovals represent state variables and dashed ovals represent derived state variables.

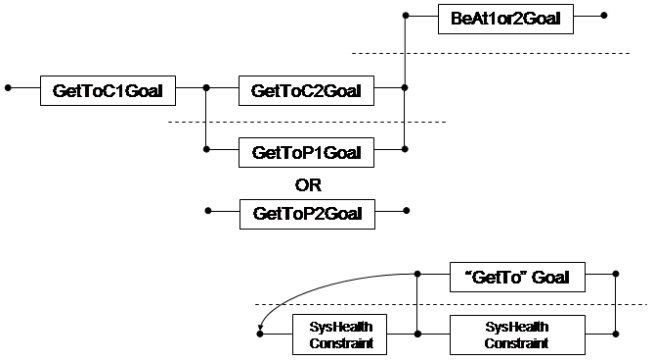

Fig. 3. Goal network and elaborations; the "GetTo" goal elaboration is relevant to the GetToC1, GetToP1, and GetToP2 goals. The dots before and after the goals are beginning and ending time points, respectively. Vertical lines between time points indicate that the time points are constrained to fire at the same time. Dashed lines under a goal indicate that the goals below it are elaborated from it.

used to estimate the robot's position, orientation, and velocity information. Several obstacles were placed in the environment to facilitate the use of the LADAR. The scan matching algorithm developed by Lu and Milios [18], which outputs position and orientation, was adapted for use in this simulation.

Several state variables are needed to describe this system. First, the position state variable tracks Cartesian and angular position and velocity, as well as the covariance matrices for the estimates. Three state variables describe the health of the three sensors as GOOD, FAIR, POOR, or FAILED. Using the same labels, the health of the overall sensing system for this specific task is described by the system health derived state variable [17]. The state effects diagram is shown in Figure 2. The health of the sensors affect the knowledge of the robot's position, and so the system health indirectly affects the knowledge of the robot's position and orientation. Since this state effect exists, it is possible for goals on the position state variable to elaborate constraints on the system health state variable.

The robot's position and orientation are estimated using a multiple model-based method [19]. In order to make the estimation algorithm robust to changes in sensor availability and health, different Kalman filters were designed for each possible combination of sensors. This approach was chosen for its relative simplicity and ease of implementation. The three sensor health variables are estimated using a different process. In each sensor's health estimator, the output of the sensor is converted to a measured position and velocity value and is compared to the other sensor's outputs. Then, a voting scheme is employed to determine the health of the sensor. Once a sensor is failed, it is assumed to always be failed. The system health derived state variable is estimated using the three sensor health state variables. The system health decays in a specific way as the sensor health values decay.

C. Goal Design

The goal network associated with this task consists of the elaboration of one overall goal, and can be seen in Figure 3. The goal is a maintenance goal on the position of the robot, called BeAt1or2Goal, which refers to locations p_1 and p_2 respectively. This goal elaborates into two goals on the robot's position, GetToC1Goal and GetToC2Goal. The first, GetToC1Goal, tells the robot to move to the first checkpoint, c_1. The second goal, GetToC2Goal, has two tactics it can elaborate; the first is GetToP1Goal and the second tactic is GetToP2Goal. These goals tell the robot to drive to the second checkpoint, which is either p_1 or p_2. The "GetTo" goals (except GetToC2Goal) elaborate goals constraining the system health state variable to be certain values. GetToP1Goal elaborates a concurrent goal constraining the system health to be GOOD and also elaborates a preceding goal that constrains the system health to be GOOD. The GetToC1Goal and GetToP2Goal both elaborate concurrent goals constraining the system health to be FAIR or better.

Initially, the GetToC2Goal elaborates to the GetToP1Goal. If the system health degrades so that it is less than GOOD before reaching the opening time point of the GetToP1Goal, the preceding system health goal will fail, causing a re-elaboration of the GetToC2Goal, which then elaborates the GetToP2Goal. However, if the system health degrades to less than GOOD while achieving the GetToP1Goal, the fault response is instead to stop the robot and go into system safing mode. The same response occurs if the system health degrades to less than FAIR while achieving the GetToC1Goal or the GetToP2Goal.

IV. Safety Verification

Hybrid system analysis tools can be used to verify the safe behavior of a hybrid system; therefore, a procedure to convert goal networks into hybrid systems is an important tool for goal network verification. The procedure described in this section allows certain structures of goal networks to be converted into simple, linear hybrid automata in a general way. There are few restrictions on the goal networks; they can constrain several state variables, which may be linearly related to each other, and they can have goals with several tactics. The amount of time needed to complete a goal can be constrained or unconstrained, and elaboration logic can be based on the state variable, affecting or affected state variables, order, and time. However, goal tactics that have constraints on controllable state variables must not introduce time points that occur during a goal on a controllable state

variable that has transition conditions that depend on completion. This type of goal is called unsplittable. Also, goals in the network must have a unique ordering or scheduling, though several goals can be active concurrently as long as the goals on the same state variable are mergeable.

The first part of the procedure is to prepare the goal network by elaborating out all tactics of the goals in the given scheduled goal network, numbering all the time points, grouping together goals that are active between consecutive time points, and labeling all state variables that are constrained in the goal network. State variables are placed into three categories: controllable, uncontrollable, and dependent. Controllable state variables (CSVs) are directly controllable and are always associated with a command class. Uncontrollable state variables (USVs) are not associated with a command class in any way. Dependent state variables (DSVs) do not have an associated command class, but have modeled dependencies on controllable state variables.

The rest of the process for converting goal networks into hybrid automata has several parts. The first describes how to create an automaton based on the control system for all the CSVs and continuous DSVs in the goal network. The next part outlines the creation of other automata based on the models of each discrete DSV and each USV. Finally, the verification of the hybrid system is the last part of the process. The process is described fully in [16], and the main points are summarized here.

For the first automaton created from the CSVs and continuous DSVs, the following steps summarize the procedure:

1) In each group, create locations (modes) by combining branch goals (goals on CSVs that are not ancestors of other goals on CSVs in the group) with all parent and sibling goals (goals in the same tactic or other root goals) that constrain CSVs. Label each location with the dynamical update equations on all CSVs and continuous DSVs constrained in the location. Create Success and Safing locations.
2) Create elaboration and transition logic tables for each goal that elaborates any constraints on CSVs and for each CSV, respectively. Outlines for these tables are shown in Table I and Table II.
3) Create transitions between locations and groups using the logic outlined in the tables from the previous step. Elaboration logic controls the transitions into groups and the failure transitions from each location, and transition logic controls the transitions out of a group to the next group or to the Success location.
4) Add exit and failure transitions based on time to locations containing goals that have time constraints. Add entry actions that reset the time variable to transitions into these locations from the group connector.
5) Remove unnecessary locations, groups, and transitions.

For discrete DSVs and all USVs, create a separate hybrid automaton for each.

1) Create a location in each automaton for each discrete state of the discrete DSV or USV used in the CSV automaton (or discrete sets of continuous states, for continuous USVs).
2) Create transitions and transition conditions between the locations that correspond to the modeled behavior of the state variable.
3) Parameterize transitions of stochastic uncontrollable state variables for verification.

Finally, for the verification of the system, each automaton must be converted into a form suitable for the verification software (which is simply a syntax issue) and transitions in the discrete DSV and USV automata and the affected transitions in the CSV automaton must be synchronized. The last step of the procedure before safety verification is establishing the "incorrect" or "unsafe" sets, which are system conditions that should never be true.

TABLE I

OUTLINE OF AN ELABORATION LOGIC TABLE

Tactic	Starts in	Fail to	Fail Conditions
1			
2			
:			

TABLE II

OUTLINE OF A TRANSITION, SUCCESS, AND FAILURE LOGIC TABLE

From	Unc	Maint	Cntl	...	Success	Fail
Unconstrained						
Maintenance						
Control						
:						

V. EXAMPLE RESULTS

Returning to the example task and goal network described in Section III, this section will describe the MDS simulation of the task as well as the safety verification of this example.

A. Simulation

The robotic simulation environment consists of three software packages. The autonomous robotic control system was implemented in MDS, and an open-source 3-D robotic simulation package called Gazebo was used to simulate the environment, the robot, and its sensors. An open-source server package called Player was used to interface between the hardware adapter in MDS and the simulated robot in Gazebo [20].

1) Results: The robotic task was simulated in a nominal case and in several sensor failure and degradation cases. Sensors were failed and degraded by intercepting the sensor measurement values supplied by Gazebo and setting them to modified values. Each of the failures and degradations were introduced at five different time points during the task.

The nominal case performed exactly as expected, as seen in Figure 4. The health of the system and sensors remains high throughout the run, and the checkpoints are achieved in order, with checkpoint c_2 occurring at p_1, which has the

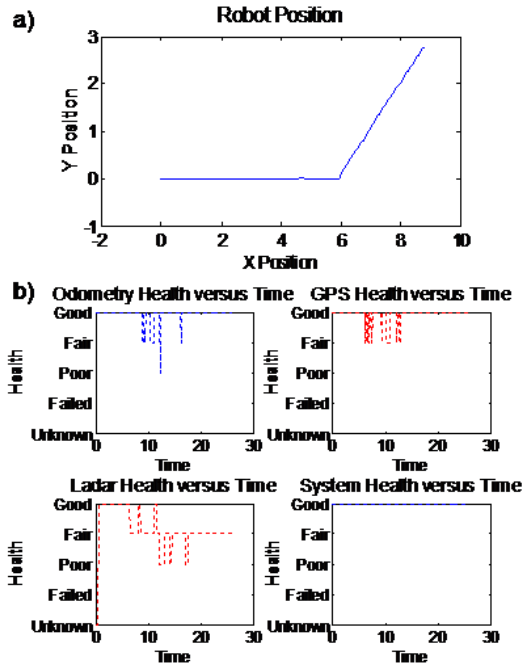

Fig. 4. Nominal case: a) position, b) health variables

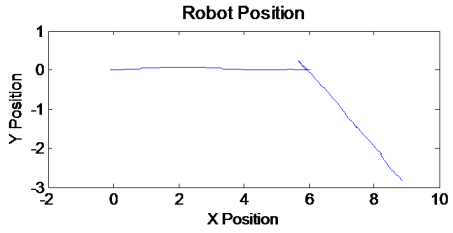

Fig. 5. Robot position in a GPS failure case

narrower error bound. The LADAR and odometry failure and degradation cases were similar to the nominal case in that each case was successful and the route to c_2 occurring at p_1 was completed in each case.

Nearly all of the GPS failure and degradation runs were completed successfully. For the runs in which the failure or degradation occurred before the GetToP1Goal became active, the re-elaboration of the GetToP2Goal was triggered and the robot completed the task by reaching p_2, as seen in Figure 5. (For one of these runs, the re-elaboration was not triggered soon enough before the transition to the next goal, and therefore, the outcome was the failure of the GetToP1Goal, causing safing). For the runs in which the failure or degradation occurred after the GetToP1Goal had started, the result was immediate goal failure and safing.

These results show that the simulation of the system successfully represents the designed behavior of the system.

B. Safety Verification

1) Hybrid System Design: Using the rules outlined in Section IV, the goal net for this example was converted to a hybrid system. The CSV automaton was developed from

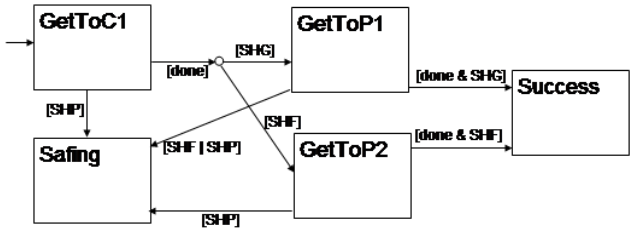

Fig. 6. Hybrid automaton for the position state variable; SHG, SHF, SHP is system health is GOOD, FAIR, and POOR, respectively; "done" indicates that the position state variable has achieved its constraint.

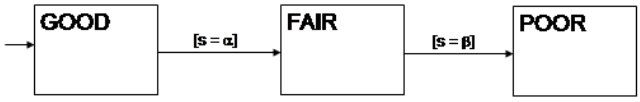

Fig. 7. Hybrid automaton for the system health state variable

the "GetTo" goals constraining the position state variable, and is represented in Figure 6. The first location entered is GetToC1; from this location, transitions to Safing (via failure logic) or to the next group (via transition logic) are possible. The second group has two locations in it, GetToP1 and GetToP2. Once the system transitions out of GetToC1, elaboration logic dictates which location (either GetToP1 or GetToP2) is entered. Transitions from either of these locations are to the Success (transition logic) or Safing (failure logic) locations.

The automaton for the USV, the system health state variable, has locations that are based on the three discrete values of the system health that affect the position state variable's automaton. As seen in Figure 7, the transition conditions are stochastic and parameterized for verification.

2) Results: HyTech, a symbolic model checking software that can analyze simple linear hybrid systems, was used to verify this system [13]. The automata for the position and system health state variables were encoded and synchronized (see Figure 8). The rates of degradation of the system health are the parameters that the safety verification search is conducted over, and these parameters are represented by α and β in Figures 8 and 7. The decay of the health of the system from GOOD to FAIR occurs when the health variable reaches a value of α and the transition from FAIR to POOR occurs when the system health variable reaches a value of β. The system health variable increases at any rate between zero and one ($\dot{s} \in [0,1]$). The rest of the model is as described above.

The initial conditions for the system are starting in the GetToC1 location for the position and GOOD for the system health. The "unsafe" set used for this analysis consists of four different regions: 1) position in Safing and health is Good; 2) position in GetToC1 and health is POOR; 3) position in GetToP1 and health is not GOOD; and 4) position in GetToP2 and health is POOR. Forward analysis from the

```
automaton health
synclabs: Fair, Poor ;
initially GOOD & s=1;

loc GOOD: while s <= alpha wait {ds in [0, 1]}
       when s >= alpha sync Fair goto FAIR ;
loc FAIR: while s<=beta wait {ds in [0, 1]}
       when s >= beta sync Poor goto POOR ;
loc POOR: while s >= beta wait {ds = 0 }
       when True goto POOR ;
end

automaton goals
initially GetToC1 & x=0 & y=0 ;
synclabs: Fair, Poor ;

loc GetToC1: while x <= 6 wait {dx in [ 1/10, 1],dy=0 }
       when x>=6 & s>=alpha & s<beta goto GetToP2 ;
       when s < alpha & x >= 6 goto GetToP1 ;
       when True sync Poor goto Safing ;
loc GetToP1: while y <= 4 wait {dx=0, dy in [1/10, 1]}
       when y >= 4 & s < alpha & s < beta goto Success ;
       when True sync Fair goto Safing ;
loc GetToP2: while y >= -4 wait {dx=0, dy in [-1, -1/10]}
       when y <= -4 & s >= alpha & s < beta goto Success;
       when True sync Poor goto Safing ;
loc Success: while True wait {dx=0,dy =0 }
loc Safing: while True wait {dx=0,dy=0 }
end
```

Fig. 8. HyTech code excerpt for position and system health automata in example problem. Note the synchronization between the decay of the system health and the transitions between position locations.

initial conditions was used. HyTech found that there are no values of α or β that would cause the system to go into any of the unsafe regions. A more complicated example that was verified using a less capable version of the conversion procedure can be found in [21].

VI. Conclusion and Future Work

This paper describes a systematic way to verify goal networks using a general procedure to translate certain types of goal networks into linear hybrid systems. A software package specializing in the analysis of linear hybrid systems can then be used to verify the safety of the system. The process was used successfully on a simple example problem, though due to the way multiple controllable and continuous dependent state variables are handled by the process, it is likely that this procedure will easily handle more complicated problems. This result is important for the development and use of reconfigurable goal networks as a method to robustly control complex embedded systems.

Future work includes the proof and automation of this procedure to translate goal networks to hybrid systems. It may also be possible to extend this procedure to apply to even more complex goal networks by using certain MDS attributes, like projections based on state models, in the transition conditions of the hybrid automata. Another extension would be to include estimation uncertainty of uncontrollable state variables in the verification procedure.

VII. Acknowledgements

The authors would like to gratefully acknowledge Kenny Meyer for his many efforts in enabling this collaborative work; a special thanks to Michel Ingham for his help with the goal net design; Robert Rasmussen, Matthew Bennett, Mark Indictor, Daniel Dvorak, and the MDS team at JPL for feedback, suggestions, answered questions, and MDS and State Analysis instruction; Jeremy Ma for supplying knowledge and code for the Lu and Milios scan matching algorithm; and Stefano Di Cairano for his help with hybrid systems, Stateflow, and HyTech. This work was funded by NSF and AFOSR.

References

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

[2] 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.

[3] 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.

[4] Z.-H. Duan, Z.-X. Cai, and J.-X. Yu, "Fault diagnosis and fault tolerant control for wheeled mobile robots under unknown environments: A survey," *IEEE Int'l Conference on Robotics and Automation*, pp. 3428–3433, 2005.

[5] C. Ferrell, "Failure recognition and fault tolerance of an autonomous robot," *Adaptive Behaviour*, vol. 2, no. 4, pp. 375–398, 1994.

[6] M. L. Visinsky, J. R. Cavallaro, and I. D. Walker, "A dynamic fault tolerance framework for remote robots," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 4, pp. 477–490, 1995.

[7] M. W. Hofbaur and B. C. Williams, "Hybrid estimation of complex systems," *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, vol. 34, no. 5, pp. 2178–2191, 2004.

[8] V. Verma, G. Gordon, R. Simmons, and S. Thrun, "Real-time fault diagnosis [robot fault diagnosis]," *IEEE Robotics and Automation Magazine*, vol. 11, no. 2, pp. 56–66, 2004.

[9] Y. Diao and K. M. Passino, "Intelligent fault-tolerant control using adaptive and learning methods," *Control Engineering Practice*, vol. 10, pp. 801–817, 2002.

[10] Y. Zhang and J. Jiang, "Fault tolerant control system design with explicit consideration of performance degradation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 39, pp. 838–848, July 2003.

[11] G. Labinaz, M. M. Bayoumi, and K. Rudie, "A survey of modeling and control of hybrid systems," *Annual Reviews of Control*, 1997.

[12] 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.

[13] 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.

[14] 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.

[15] D. Dill and H. Wong-Toi, *CAV 95: Computer-aided Verification*, ch. Verification of real-time systems by successive over and under approximation, pp. 409–422. Springer, 1995.

[16] J. M. Braman and R. M. Murray, "Conversion and verification procedure for goal-based control programs," tech. rep., California Institute of Technology, 2007. CaltechCDSTR:2007.001, http://caltechcdstr.library.caltech.edu/view/.

[17] D. Dvorak, R. Rasmussen, and T. Starbird, "State knowledge representation in the Mission Data System," *IEEE Aerospace Conference*, 2002.

[18] F. Lu and E. Milios, "Robot pose estimation in unknown environments by matching 2D range scans," *Journal of Intelligent and Robotic Systems*, vol. 20, pp. 249–275, 1997.

[19] L. Drolet, F. Michaud, and J. Côté, "Adaptable sensor fusion using multiple Kalman filters," *IEEE Int'l Conference on Intelligent Robots and Systems*, vol. 2, pp. 1434–1439, 2000.

[20] "The Player project." http://playerstage.sourceforge.net/, November 2006.

[21] J. M. Braman, R. M. Murray, and M. D. Ingham, "Verification procedure for generalized goal-based control programs," *AIAA Infotech@Aerospace*, 2007.