# Detection and Prediction of Adverse and Anomalous Events in Medical Robots

**Kai Liang, Feng Cao, Zhuofu Bai, Mark Renfrew, M. Cenk Çavuşoğlu, Andy Podgurski, Soumya Ray**
Department of Electrical Engineering and Computer Science
Case Western Reserve University, Cleveland, OH 44106, USA
{kxl307, fxc100, zxb31, mark.renfrew, mcc14, podgurski, sray}@case.edu

## Abstract

Adverse and anomalous (A&A) events are a serious concern in medical robots. We describe a system that can rapidly detect such events and predict their occurrence. As part of this system, we describe simulation, data collection and user interface tools we build for a robot for small animal biopsies. The data we collect consists of both the hardware state of the robot and variables in the software controller. We use this data to train dynamic Bayesian network models of the joint hardware-software state-space dynamics of the robot. Our empirical evaluation shows that (i) our models can accurately model normal behavior of the robot, (ii) they can rapidly detect anomalous behavior once it starts, (iii) they can accurately predict a future A&A event within a time window of it starting and (iv) the use of additional software variables beyond the hardware state of the robot is important in being able to detect and predict certain kinds of events.

## Introduction

*Medical robotic systems* are cyberphysical systems that are used to plan and perform medical interventions with high precision and repeatability (stereotactic surgery), to allow access to places and scales that are not accessible with manual instruments and conventional techniques (microsurgery at small scales and minimally invasive surgery where access is limited), or to perform surgery with the use of large amounts of quantitative information (image-guided surgery) (Taylor and Stoianovici 2003; Çavuşoğlu 2006). These systems can improve patient health and reduce costs by ensuring precision and accuracy, thereby decreasing time in the operating room, speeding patient recovery time and minimizing side effects.

An advanced medical robotic system is quite complex, with regard to both its electromechanical design and the software that provides its user-interface, coordinates its activities, and controls the system's actuators. This complexity increases the risk of dangerous accidents due to hardware and software malfunctions, observability limitations, or human-machine interface problems. Indeed such events have already occurred, as evidenced by a number of adverse event reports filed by manufacturers with the Food and Drug Administration (FDA). One such report (FDA 2008) contains the following description of an event involving the da Vinci S Surgical System (Intuitive Surgical Inc. 2009):

> *[D]uring a da Vinci's beating heart double vessel coronary artery bypass graft procedure at the hospital, there was an unexplained movement on the system arm which had the endowrist stabilizer instrument attached to it. [This] caused the feet at the distal end of the endowrist stabilizer instrument to tip downward resulting in damage to the myocardium of the patient's left ventricle.*

The accompanying "Manufacturer Narrative" states:

> *The investigation conducted by an isu field service engineer found the system to [have] successfully completed all verification tests and to function as designed. No system errors related to the cause of this event were found. Additional investigations conducted by clinical and software engineering were inconclusive as a root cause could not be determined based on the event descriptions reported by several eye witnesses and the review of the system's event logs.*

In this paper we describe a prototype system to enhance the safety of medical robotic systems by monitoring their behavior and detecting or even predicting where possible such *adverse and anomalous* (A&A) events. This system is built and evaluated using a simulation of a robot that we are concurrently developing for small-animal biopsies. While work exists that attempts to *design* safe systems and validate their behavior before deployment, completely safe behavior is difficult to guarantee in practice. Our approach is meant to *augment* approaches such as FMAE analysis, model checking, etc. that are applied during the design process by taking into account how a medical robotic system behaves in the field and how its observable behavior is related to its hardware and software dynamics. This behavior can also be a function of how clinicians employ it, how the system affects patients and how it is affected by operating conditions.

Effectively detecting, predicting and responding A&A events in an online manner requires the eventual solution of several key subproblems: (1) devising efficient means of collecting pertinent hardware and software execution data, as well as user feedback; (2) developing statistical learning models to effectively relate the collected data to the occurrence of A&A events; (3) designing a robust simulation

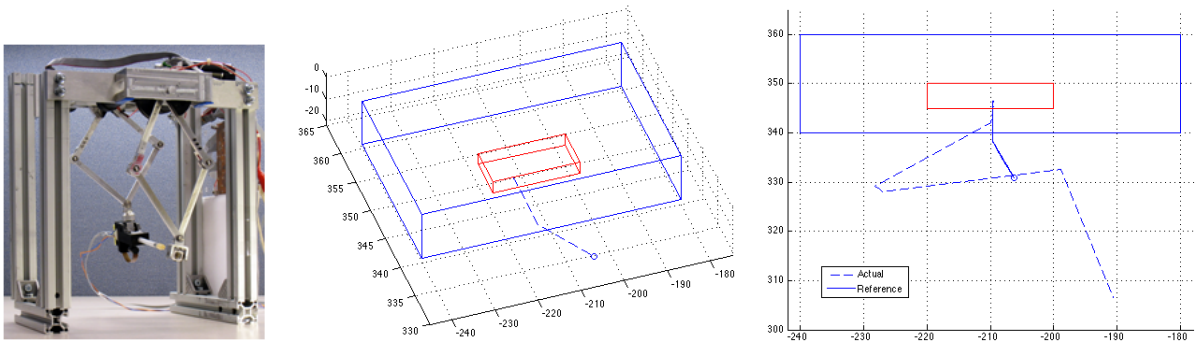


Figure 1: The SABiR robot (left), a normal needle-tip trajectory in the simulation environment consisting of two "tissue" blocks (center), and a trajectory with an A&A event caused by encoder failure (right) showing the reference and actual trajectories.

platform to reproduce observed behavior and (4) developing clinically effective response strategies. This paper describes approaches that address the first three subproblems. We have built a detailed simulator of the robotic system we use and a supervised software controller and GUI, instrumented for data collection. A novel aspect of our approach is that we collect *both* software and hardware data during operation. Using the collected data, we learn dynamic Bayesian network (DBN) models of the system's behavior. When learning, we use the simulator of the robot as an oracle to ensure we obtain accurate models. In our evaluation, we show that the models we learn are able to effectively detect A&A events and also predict them in certain cases.

## System Description

In this section, we describe our testbed robot, the simulation environment, supervisory control software and GUI and the statistical models we learn from the collected data.

**The SABiR Robot.** In our work, we use the Small Animal Biopsy Robot (SABiR), designed and built in our lab (Bebek et al. 2008). Figure 1 (left) shows an image of the robot. It is a five-degree-of-freedom parallel robotic manipulator which is designed to take biopsies or deliver therapeutic drugs at targets in live small animal subjects and to achieve accuracy better than 250$\mu$m. It employs a parallel design to achieve low inertia. The robot has high position resolution and can realize dexterous alignment of the needle before insertion. The design is lightweight and has high motion bandwidth, so that biological motion (e.g., breathing, heartbeat, etc) can be canceled while the needle is inserted in tissue.

The robot consists of a needle mechanism held by two 5-bar linkage mechanisms, called the front and rear stages. The front stage has two degrees of freedom (up/down, left/right) and the rear stage has three degrees of freedom (up/down, left/right, rotate forward/rotate backward). The stages are driven by five tendon mechanism motors and the joint angles are measured by encoders. The robot's hardware state is characterized by its five joint angles, and there is a one-to-one correspondence between any position and orientation that the needle tip can reach and the set of joint angles.

**Robot Simulation and Environment.** We use models for the kinematics and inverse kinematics developed in our prior work (Hwang et al. 2009) to create a simulation of the robot, implemented in Simulink, in which the robot's motors are each represented as fifth-order transfer functions. The simulator is designed to be a modular component, so that it can be seamlessly swapped with the controller of the actual robot.

The environment of the simulated robot consists of a gel block (to simulate "tissue") placed in the workspace (Figure 1 (center)). A needle force model, which assumes a stiff non-deformable needle, is used to provide a resistive force caused by the combined frictional, cutting, and stiffness forces produced when the needle is inside the gel block. The cutting force is caused by the needletip piercing the gel block and provides a resistance to the needle's motion during insertion into the gel block. The frictional force is produced by the friction between the needle body and the walls of the channel in the gel block, and resists the needle during insertion and extraction. The stiffness force is caused by the gel block's tendency to resist sideways motion of the needle, i.e., any motion not in the direction the needle is pointing. In this way, realistic and distinguishable forces can be produced by any possible motion of the needle. The needle model is described in detail in (Jackson and Çavuşoğlu 2012).

After calibration the robot's initial location is called its "home" point. The controller can then be given new points and orientations to move the needle to. A "reference" trajectory is computed using linear interpolation. Velocities along this trajectory are set to be mainly constant, with fast acceleration and deceleration at the start and end (subject to a desired maximum acceleration). We then use a two-layer motion control algorithm to follow this trajectory to guide the needle to the end point. The inner layer is a pole-placement based joint-level position controller operation at a sampling rate of 2 kHz. The outer layer is a cartesian-space position controller operating at 100 Hz.

**Supervisory Software.** The user interacts with the robot through a supervisory software system built on top of the low-level controller. This system has three components: a GUI, a task delegator, and a robot proxy.

The software has a graphical user interface (GUI) that allows a user to view the current robot state and specify high level actions such as "insert needle." For each such com-

Table 1: State variables in DBNs. First group: parameters, second group: software, third group: hardware.

| Description | No. |
|---|---|
| Reference position, starting position, end position | 18 |
| Position and orientation where insertion begins | 6 |
| Insert distance into tissue from ready position | 1 |
| High level action, e.g. "Insert Needle" | 2 |
| Estimated depth of the needle inside tissue, using estimated forces | 1 |
| Estimated depth of the needle inside tissue, assuming fixed tissue geometry | 1 |
| Estimated force on the needle | 3 |
| Estimated torque on the needle | 3 |
| Intermediate computation results in controller software | 87 |
| Position and orientation of needle tip | 6 |
| Positions of 5 motors | 5 |
| Torques of 5 motors | 5 |
| Error between Actual Position and Reference Position | 5 |
| Speeds of 5 motors | 5 |
| Error between Actual Speed and Reference Speed | 5 |

mand, the GUI then lists all parameters whose values need to be input or adjusted. The data is then sent to a "task delegator" component. This first checks the validity of input parameters for the specified operation; for example, it ensures that target locations are within the robot's workspace. It then decomposes a complex task into a set of basic needle motions that can be accomplished by calls to the API for the robot (or the simulator). The delegator is equipped with different schemas to decompose different high level tasks. It generates reference trajectories for the tasks and invokes the robot API to execute these tasks. As the task is being executed, it updates the robot's state on the GUI. If an error occurs, it is responsible for stopping the current action and alerting the user. The last stage, the "robot proxy," handles communications with the robot (or simulation) and low-level operations and collects low-level sensor data. This ensures that when the simulation is replaced by the actual robot, only the last stage changes. A final point is that we designed the entire architecture to be easy to monitor and log, so that we can monitor the system as it executes and collect the hardware/software data we need.

**Modeling the State Dynamics.** We use dynamic Bayesian networks (DBNs) (Dean and Kanazawa 1990) to model the time-evolution of the state space of the system based on the collected data. These are first order Markov models that represent the probability of the next state given the current one, i.e. $\Pr(S_{t+1}|S_t)$, where each $S_t$ is described by a vector of variables, as follows:

$$\Pr(S_{t+1}|S_t) = \Pr(\mathbf{V}_{t+1}|\mathbf{V}_t) = \prod_{i=1}^{n} \Pr(V_{t+1}^i|\mathbf{V}_t),$$

where $\mathbf{V}_t = \{V_t^i\}$ denotes all of the variables in the $t^{th}$ time step. The structure and parameters of these probability distributions are learned from data as described below.

We specify the conditional probability distributions (CPDs) in the DBN in two ways. Certain state variables, such as the $x$ position of the needle tip, vary at a near linear rate from $t$ to $t+1$, for example because (in this case) the robot controllers are designed to maintain a constant velocity as far as possible. For such variables, we use linear Gaussian models, so for example $X_{t+1}|\mathbf{V}_t \sim N(\mathbf{w}_X \cdot \mathbf{V}_t, \sigma_X)$. For other variables, we employ a regression tree model (Breiman et al. 1984) for the CPD. Each internal node of the regression tree is a test on some variable at the previous time step. Each leaf node is again a linear Gaussian model. Regression tree models such as these are a very general representation of nonlinear dynamics.

Since this is an engineered system, it is designed to be sparse, i.e. most state variables tend not to depend on too many other variables. Thus, when learning the regression tree/linear Gaussian CPDs, we use the Sparse Candidate (Friedman, Nachman, and Pe'er 1999) feature selection algorithm to limit the number of parents for each variable. Given a current structure, this algorithm first computes a candidate parent set for each variable limited to a certain size specified in the input, and then learns a structure with these sets. These two steps are iterated until convergence.

There are three types of variables that are part of the system state. Variables such as the reference trajectory to be followed by the robot are "parameters" that do not change over time. They help predict other variables, but are not themselves predicted. Other variables such as the motor torques are "hardware variables." These variables are obtained by sensors on the robot, or through direct computations from these measurements. The third set of variables are "software variables." These variables include flags denoting which high level motion is being executed, which are set in the software and variables such as "force on the needle" which cannot be directly sensed in the hardware but can be *estimated* in software indirectly from other variables. They also include variables from the controller software that store intermediate computation results. These are included because A&A events may originate not just from hardware malfunctions but errors in the controller code as well. The full set of variables is shown in Table 1. For each kind of variable, "No." refers to the number of variables of that kind, e.g. there are 3 needle forces, one in each direction.

Since it is impossible to know ahead of time what sort of A&A events to expect, we learn DBNs to model "normal" state transitions. To do this we generate sequences of normal trajectories from our simulation and estimate the CPDs for these variables from them. CPD parameters are estimated using maximum likelihood; for linear Gaussian models, this is equivalent to linear regression and yields closed form solutions. For regression tree models, we use the standard greedy top down recursive decomposition approach (Breiman et al. 1984), where the goodness of a split is computed by the improvement in the $r^2$ measure. We make one modification to this tree construction procedure. Normally, the number of datapoints decreases deeper in the tree because of recursive partitioning. However, since we have a simulator, we use the simulator as an oracle to generate datapoints as needed. These points are generated using a form of rejection sampling; a random trajectory is sampled and a point on it is evaluated to see if it satisfies the checks at the

Table 2: Average test $r^2$ for normal state trajectories.

| Model | Hardware | Software | All |
|---|---|---|---|
| HS | 1.000 | 0.9784 | 0.9817 |
| HS10 | 1.000 | 0.9582 | 0.9654 |
| HWOnly | 0.9996 | N/A | N/A |

Table 3: Detection and Prediction times (ms) at FPR=0.06.

| Model | Detection | | Prediction, Sweep | |
|---|---|---|---|---|
| | Encoder | Sweep | $k$=1000 | $k$=2000 |
| HS | 1 | - | 1518 | - |
| HS10 | 10 | 10 | 5136 | 4614 |
| HWOnly | 1 | - | 1526 | 1899 |

internal nodes. If so, it is kept, or else discarded. This procedure ensures that we have enough points at each node in the tree to make good decisions about the choice of splits. As we show in the following section, this approach yields very accurate models of the system dynamics.

## Empirical Evaluation

In this section we evaluate how well our models represent the robot, and how accurately they can detect and predict A&A events of certain types. We perform these experiments with our simulator, which is an accurate simulation of the robot. In these experiments, the simulation environment is set up with two blocks of "tissue" of different characteristics, one contained within the other (Figure 1 center). The task for the robot is to insert the needle tip a specified distance within the tissue. We consider three kinds of A&A events. The first is an "encoder failure" event, where at some point within the trajectory, the hardware element reporting a motor's position is lost, so the system can longer track that motor's position (Figure 1 right). The second is a "sweep" event, where prior to needle insertion, the needle tip strays and grazes the tissue surface. The third is an "out-of-workspace" (OOW) event, where a check for an illegal target position that is outside the robot's workspace is missing in the controller software, and such a target is input at runtime. We generate trajectories for each such event. Since actual A&A events are rare, we restrict the proportion of such "A&A data" in our datasets to 1.25%. We evaluate three DBNs: a model using all the variables in Table 1 (HS), a model using only the "parameter" and "hardware" variables (HWOnly) and a model using all variables but making 10-step predictions (i.e. modeling $\Pr(S_{t+10}|S_t)$) (HS10).

**Modeling normal trajectories.** We first evaluate how well our DBNs can model normal trajectories. We generate a test set of 400 normal trajectories and sample $20,000$ $(s_t, s_{t+1})$ pairs from them. Using the $s_t$ values, we use the DBNs to predict $\hat{s}_{t+1}$ and compute an $r^2$ metric measuring the accuracy of these predictions. From the results, shown in Table 2, we observe that the DBNs can perfectly capture the time-evolution of the hardware variables. It is more difficult to predict the software variables, however, as we show below, the accuracy is high enough to allow us to detect and predict A&A events. Finally, we observe that when using the HWOnly model, the accuracy of prediction on the hardware variables is marginally less than when using the HS model. Although the difference is very small, this suggests that the software variables add value to the DBN, and using them results in more accurate predictions. This is substantiated further in the results below.

**Detecting A&A events.** Next we consider how quickly our models can detect A&A events after the event has occurred. Of course, we would like not just to detect these events but to predict them, and we discuss prediction below. However, some A&A events, such as our encoder failures, may be unpredictable in that the trajectory appears completely normal until the point when the event happens. Therefore it is still valuable to ask, given that an A&A event has happened, how quickly a model such as we use detects it. To measure this we use a test set of 400 normal trajectories and 5 trajectories for each of the A&A events. In this case, the "ground truth" is set as follows: every point after an A&A event until the end of the event receives a label "positive," while every other is labeled "negative." We use the DBNs to check every $(s_t, s_{t+1})$ and associate each point with a smoothed negative log likelihood (NLL) score, where the smoothing is done over a window of 50 previous time steps. The smoothing helps to reduce error in intermediate short regions where the DBN's estimate is poor. We then use the smoothed NLL score to construct an ROC graph, shown in the top row of Figure 2. Further, for FPR=0.06, we also compute the average time-to-detect an A&A event, by finding the first point after an A&A event that exceeds the associated threshold. These times-to-detect are shown in Table 3 (lower is better).

From these results we observe that while all the models are good at detecting encoder failures, the HS10 models have an advantage. This is probably due to the built-in "lookahead" in these models. For the sweep event, we observe that the HS10 and HS models are more accurate than HWOnly. This is likely because the sweep event is easier to detect with software variables than hardware variables alone; when the needle grazes the tissue, the software variable "estimated depth" becomes nonzero when it should be zero in a normal trajectory. For the OOW event, it is interesting that though it is caused by a software bug, HWOnly detects it quickly. This is likely because this bug produces very bad behavior in our controller, and causes the needle to move wildly, which is easily detectable by HWOnly. Finally, from Table 3 (left) we observe that all models can quickly detect encoder failures (within 1-10ms after it happens) [1], and HS10 also quickly detects sweep events. (The other models do not detect any sweep events at FPR=0.06.)

**Predicting A&A events.** Finally, we consider the task of predicting A&A events. Imagine that the model is being used in an online setting where at every step it can make a determination as to whether an A&A event is likely to occur in the next $k$ steps. To do this, at every point $t$, we obtain

[1] For the "-" entries, in each case, only one event trajectory was correctly identified at FPR=0.06. These do not produce reliable estimates, so we have left them out.

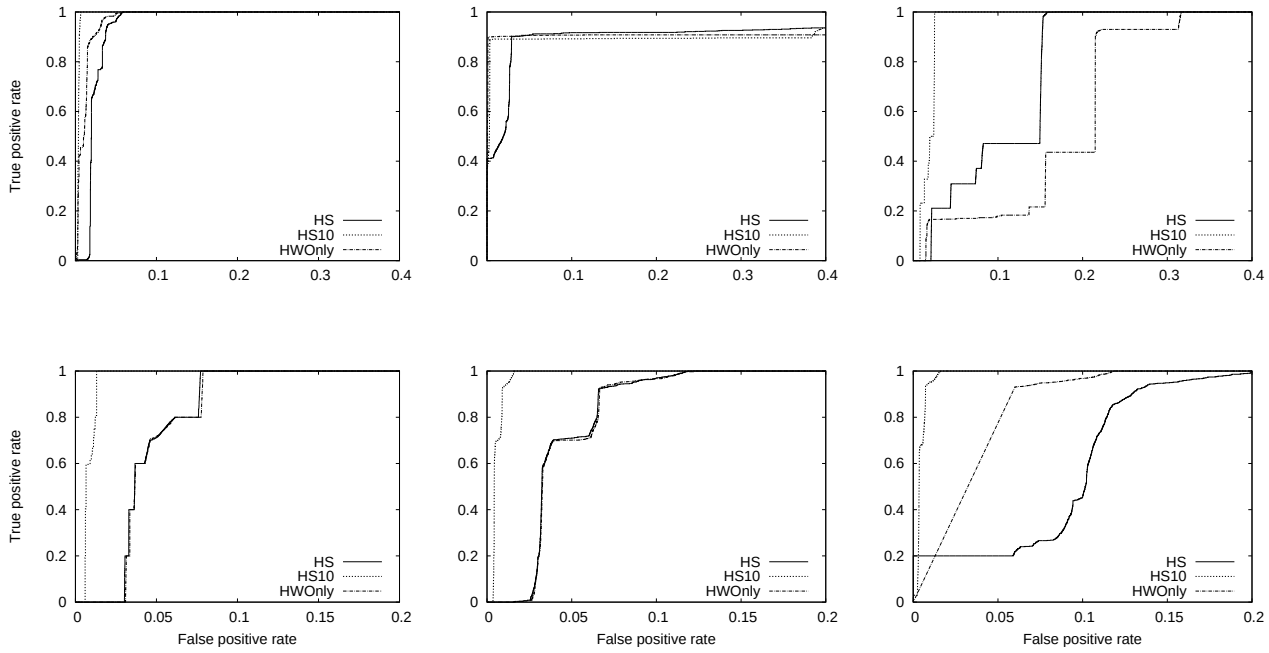


Figure 2: ROC graphs for detection and prediction of A&A events. The $x$-axes are truncated to better illustrate differences at low false positive rates. **Top row**: Detection of A&A events. Left: OOW, middle: encoder failure, right: sweep. **Bottom row:** Prediction of sweep. Left: $k = 100$, middle: $k = 1000$, right: $k = 2000$.

$k$ points $s_{t+1}, \ldots, s_{t+k}$ from our DBNs conditioned on $s_t$. These $k$ points are the means of the associated Gaussian distributions, so this is the most likely trajectory conditioned on $s_t$. From these points we pick the needle tip positions and measure the average NLL score compared to the *reference* trajectory provided as input. (We use only the needle tip position because that is the only information in the reference trajectory.) Thus we are evaluating, given the current state, how likely it is that the most likely needle tip trajectory $k$ steps later will be significantly different from the reference. Each $s_t$ is then associated with this average NLL score. The ground truth for each $s_t$ is as follows: if an A&A event does happen within $k$ steps, it is labeled "positive," else negative. We then construct an ROC graph from these predictions. In the bottom row of Figure 2, we show the results for $k = 100$ (0.05s), $k = 1000$ (0.5s) and $k = 2000$ (1s) for the sweep event. We also measure the time-to-event (Table 3 right). This is the average time between when the event was first predicted to when it happens, at a threshold corresponding to an FPR of $0.06$. Higher is better.

From these results, we observe that HS10 does very well, closely followed by both HS and HWOnly. HWOnly is comparable to HS and even better for large $k$. This may be because in this scenario, only (a subset of) the hardware variables are evaluated to calculate the NLL scores (because the reference trajectory has only those variables). As a result, the advantage of predicting the software variables may be limited in this setting, and it is likely that most of the benefit of HS10 comes from the lookahead. From Table 3 we see that the HS10 model is also able to predict this A&A event much earlier than the other models.

To summarize, all of the DBNs we test perform well in our experiments; however, the DBN that models both hardware and software variables and also looks ahead, HS10, consistently works better at both detection and prediction. These results also indicate that DBNs of the type we use can be successful in identifying A&A events and modeling elements of the software state can be helpful in classification and detection of these events.

## Related Work

Most related research on the safety of medical robotic systems in the literature primarily focus on design of intrinsically safe systems, e.g. (Taylor and Stoianovici 2003; Davies 1996; Dombre et al. 2001; Duchemin et al. 2004; Ellenby 1994). A related approach in hybrid systems is parameter synthesis (Donzé, Krogh, and Rajhans 2009). Here system parameters, such as joint limits, power levels, mass, etc. are designed in such a way as to produce good behavior and minimize or eliminate risk and/or the system is designed to fail in a safe manner and come to a controlled halt so that it can be removed and the procedure completed manually. This is typically achieved by using actuators with limited power and speed, current limiters, etc. There are also studies which lay out approaches based on identification of potential hazards and mitigating them throughout the development

lifecycle using hazard analysis and formal methods (Jetley, Iyer, and Jones 2006; Fei et al. 2001; Hu et al. 2007; Varley 1999). These approaches are generally complementary to ours, which uses statistical learning methods to analyze observed behavior.

Online fault detection and diagnosis is a very well studied problem in general robotics and other hybrid systems (e.g. (Halder and Sarkar 2007; McIntyre et al. 2005; Verma et al. 2004)). A common approach is to use probabilistic sequence models to represent the system and to perform online inference to detect when the system is in a faulty state. These models typically focus on modeling the hardware and devote attention to efficient inference algorithms to account for the online setting. Recent work on diagnosis has started to look at software as well (Mikaelian, Williams, and Sachenbacher 2005). Our work also uses statistical models, but is different in that (i) our models are learned using a simulation of the system as an oracle, (ii) we focus primarily on A&A events in medical robots and (iii) we consider the joint hardware-software state of the system in our models.

In software engineering, prior work has analyzed safety-critical systems, such as spacecraft (Lutz and Mikulski 2003), and recommended the use of runtime monitoring to detect faults. Unlike our work, this work is typically not in the context of robotic systems or medical robots, however.

## Conclusion

In this paper, we have described an approach to detecting and predicting A&A events in medical robots. Our results indicate that the approach is capable of accurately modeling our testbed robot. We are currently concurrently improving the simulation and controller by adding path planning and a simulation of image-guidance and improving the software architecture to enable more complex high level actions to be performed and to collect more data about the software execution. In the modeling framework, we are working on active-learning framework for more efficient training.

## Acknowledgments

This work is supported in part by NSF CNS-1035602. Ray is supported in part by CWRU award OSA110264.

## References

Bebek, O.; Hwang, M. J.; Fei, B.; and Çavuşoğlu, M. C. 2008. Design of a small animal biopsy robot. In *30th Intl. Conf. of the IEEE Engg. in Medicine and Biology Soc.*, 5601–5604.

Breiman, L.; Friedman, J.; Olshen, R.; and Stone, C. 1984. *Classification and Regression Trees*. Wadsworth and Brooks.

Çavuşoğlu, M. C. 2006. *Wiley Encyclopedia of Biomedical Engineering*. John Wiley and Sons, Inc. chapter Medical Robotics in Surgery. M. Akay, editor.

Davies, B. L. 1996. *Computer-Integrated Surgery: Technology and Clinical Applications*. R. H. Taylor et al., ed.s, 287–300.

Dean, T., and Kanazawa, K. 1990. A model for reasoning about persistence and causation. *Comput. Intell.* 5(3):142–150.

Dombre, E.; Poignet, P.; Pierrot, F.; Duchemin, G.; and Urbain, L. 2001. Intrinsically safe active robotic systems for medical applications. In *Proceedings of the* $1^{st}$ *IARP/IEEE-RAS Joint Workshop on Technical Challenge for Dependable Robots in Human Environments*.

Donzé, A.; Krogh, B.; and Rajhans, A. 2009. Parameter synthesis for hybrid systems with an application to Simulink models. In *12th International Conference on Hybrid Systems: Computation and Control*, 165–179. Springer-Verlag.

Duchemin, G.; Poignet, P.; Dombre, E.; and Peirrot, F. 2004. Medically safe and sound [human-friendly robot dependability]. *Robotics & Automation, IEEE* 11(2):46–55.

Ellenby, S. B. 1994. Safety issues concerning medical robotics. In *IEEE Colloquium On Safety and Reliability of Complex Robotic Systems*, 3–10. IET.

FDA. 2008. Adverse event report 2955842-2008-01144: Intuitive surgical inc., Da Vinci S Surgical System endoscopic instrument control system.

Fei, B.; Ng, W. S.; Chauhan, S.; and Kwoh, C. 2001. The safety issues of medical robotics. *Reliability Engineering & System Safety* 73(2):183–192.

Friedman, N.; Nachman, I.; and Pe'er, D. 1999. Learning of Bayesian network structure from massive datasets: The "sparse candidate" algorithm. In *Fifteenth Conference on Uncertainty in Artificial Intelligence*. Stockholm, Sweden.

Halder, B., and Sarkar, N. 2007. Robust fault detection of a robotic manipulator. *Int. J. Robotics Research* 26(3):273–285.

Hu, Y.; Podder, T.; Buzurovic, I.; Yan, K.; Ng, W.; and Yu, Y. 2007. Hazard analysis of EUCLIDIAN: An image-guided robotic brachytherapy system. In *29th IEEE Engineering in Medicine and Biology Society (EMBS)*.

Hwang, M. J.; Bebek, O.; Liang, F.; Fei, B.; and Çavuşoğlu, M. C. 2009. Kinematic calibration of a parallel robot for small animal biopsies. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 4104–4109.

Intuitive Surgical Inc. 2009. Da Vinci S Surgical System.

Jackson, R. C., and Çavuşoğlu, M. C. 2012. Modeling of needle-tissue interaction forces during surgical suturing. In *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

Jetley, R.; Iyer, S. P.; and Jones, P. 2006. A formal methods approach to medical device review. *Computer* 39(4):61–67.

Lutz, R. R., and Mikulski, I. C. 2003. Operational anomalies as a cause of safety-critical requirements evolution. *Journal of Systems and Software* 65(2):155–161.

McIntyre, M. L.; Dixon, W. E.; Dawson, D. M.; and Walker, I. D. 2005. Fault identification for robot manipulators. *IEEE Transactions on Robotics* 21(5):1028–1034.

Mikaelian, T.; Williams, B.; and Sachenbacher, M. 2005. Model-based monitoring and diagnosis of systems with software-extended behavior. In *Proc.* $20^{th}$ *Natl. Conf. on AI*.

Taylor, R. H., and Stoianovici, D. 2003. Medical robotics in computer integrated surgery. *IEEE Transactions on Robotics and Automation* 19(5):765–781.

Varley, P. 1999. Techniques for development of safety-related software for surgical robots. *IEEE Transactions on Information Technology in Biomedicine* 3(4):261–267.

Verma, V.; Gordon, G.; Simmons, R.; and Thrun, S. 2004. Real-time fault diagnosis. *IEEE Robotics & Automat.* 11(2):56–66.