A Framework for Detecting Adverse and Anomalous Events in Medical Robotic Systems

Feng Cao[1], Mark Renfrew, Zhuofu Bai, M. Cenk Çavuşoğlu, Andy Podgurski and Soumya Ray

***Abstract*— We propose a framework to model and evaluate the reliability of complex cyberphysical systems like medical robots. As part of this framework, we describe data collection, accurate simulation and user interface tools we build for a robot for small animal biopsies. The data we collect consists of relevant variables from the software and hardware state of the robot. We use this data to construct dynamic Bayesian network models of the software/hardware state space of the system, and then use these models to detect adverse and anomalous (A&A) events if they occur. Our empirical evaluation shows that (i) our models can accurately capture the hardware/software dynamics, (ii) they can detect and also predict certain kinds of A&A events and (iii) modeling both hardware and software state for this purpose is often better than using the hardware state alone.**

I. Introduction

Medical robotic (MR) 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) [1], [2]. 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, in regard to both its electromechanical design and the software that provides its user-interface, coordinates it 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 (HMI) 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 [3] contains the following description of an event involving the da Vinci S Surgical System [4]: *It was reported that during 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. The unexpected movement 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.*

Our goal is to enhance the reliability and safety of medical robotic systems by developing a framework for empirically assessing and monitoring these properties during simulation, field testing, and general deployment of a system. This framework, illustrated in Figure 1, is intended to augment forms of validation that are applied earlier (e.g., FMAE analysis, model checking, etc.), by taking into account how a medical robotic system behaves in the field, how clinicians employ it, how the system affects patients, how it is affected by operating conditions, and how its observable behavior is related to its hardware and software dynamics. In this work we describe a limited prototype of the framework we build and evaluate using a simulation of a robot that we are concurrently developing for small-animal biopsies.

A major goal of our research is to effectively predict, detect, and respond to adverse and anomalous system events (A&A events) that threaten patient safety. This in turn 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 sim-

[1] Contact author (feng.cao@case.edu). All authors are with the Department of Electrical Engineering and Computer Science, Case Western Reserve University, Cleveland, OH 44106, USA.

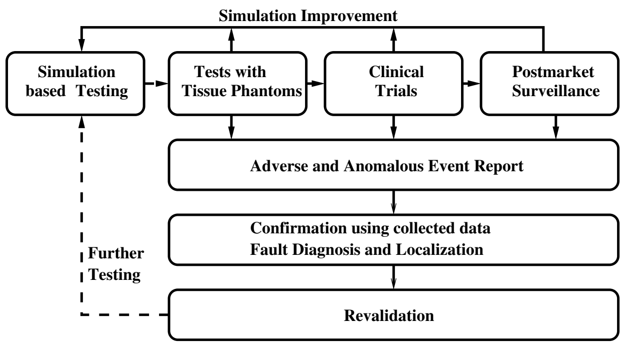
Simulation Improvement
Simulation based Testing
Tests with Tissue Phantoms
Clinical Trials
Postmarket Surveillance
Adverse and Anomalous Event Report
Confirmation using collected data
Fault Diagnosis and Localization
Further Testing
Revalidation

Fig. 1. A "phased" validation framework for medical robotic systems.

ulation 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 simulator of the robotic system we use and an associated high level software controller and GUI, instrumented for data collection. Using the collected data, we build dynamic Bayesian network (DBN) models of the system's behavior. In our evaluation, we test the ability of these models to classify, detect and predict A&A events. Our results are encouraging and indicate that our approach provides a suitable basis for the proposed framework.

II. Related Work

Two bodies of work are most related to our own: prior work on the safety of medical robotic systems and the literature on parameter synthesis, detection and diagnosis of failure in general robots. Most research on the safety of medical robotic systems in the literature primarily focus on design of intrinsically safe systems [1], [5], [6], [7], [8], [9], [10]. A related approach in hybrid systems is parameter synthesis [11]. 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 or the system is designed to fail in a safe manner and come to a controlled halt so that it can be removed and procedure completed manually. This is typically achieved by using actuators with limited power and speed, current limiters, redundant sensors, safety monitors at the sensor, servo and supervisory levels, and watchdogs monitoring activity of the control system. 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 [12], [13], [14], [15]. These approaches are generally complementary to our approach, which relies on statistical 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. [16], [17], [18], [19]). 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 [20]. Our work also uses statistical models, but is different in that we focus primarily on medical robots and consider both online and offline classification and prediction scenarios in this setting.

Related work also exists in software engineering. Prior work has analyzed safety-critical systems, such as spacecraft [21], and recommended the use of runtime monitoring to detect faults. There is also a large body of work that uses probabilistic models for software testing and fault localization (e.g. [22], [23]). Unlike our work, this work is typically not in the context of robotic systems or medical robots, however.

III. Testing Framework

The overall testing framework we propose is shown in Figure 1. This is a phased validation strategy where the device is tested in several ways: first through accurate simulations, then in hardware with tissue "phantoms" that simulate real tissue, then in clinical trials and finally when it is marketed. At each point data is collected and analyzed to determine possible A&A events. Reports of A&A events are also collected from users. If confirmed, diagnosis and localization techniques can be used to identify and correct the malfunctioning hardware/software. The modified system is then revalidated using the same phased validation pipeline.

In this paper, we focus on the simulation-based testing phase. We develop an accurate simulation of our robot, the associated software and statistical methods to analyze the collected data. In the following we describe each of these in detail. In the next section, we describe experiments to evaluate how accurately our models can detect A&A events in our environment.

A. The SABiR Robot

In our work, we use the Small Animal Biopsy Robot (SABiR), designed and built in our lab [24]. Figure 2 (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 250μm. It employs a parallel design to achieve low inertia. The robot has high position resolution and can

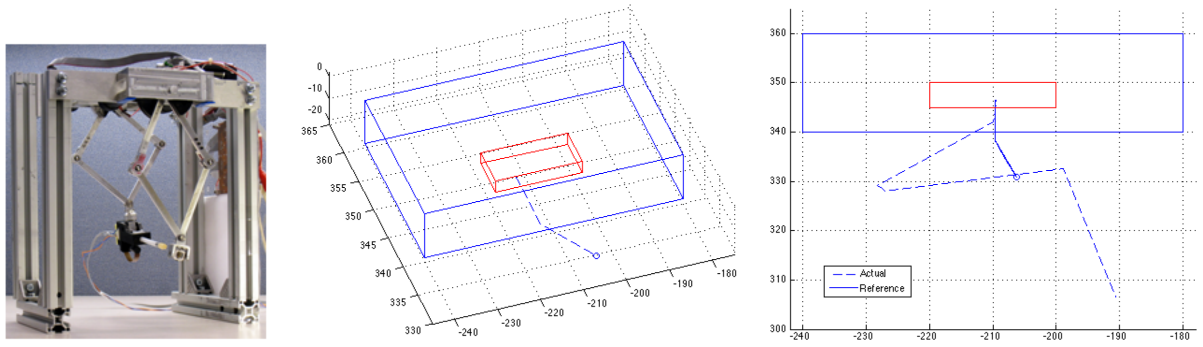

Fig. 2. The SABiR robot (left), a normal trajectory in the simulation environment consisting of two “tissue” blocks (center), and a trajectory with an A&A event caused by encoder failure (right) showing the reference and actual trajectories.

realize dexterous alignment of the needle before insertion. The design is light weight, and has high motion bandwidth, so that biological motion (e.g., breathing, heartbeat, etc) at the target can be canceled while the needle is inserted inside tissue.

The robot consists of a needle mechanism held by two 5-bar linkage mechanisms, referred to as 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 state is characterized by its five joint angles, and there is a one-to-one correspondence between any position and orientation that the needletip can reach and a set of joint angles.

B. Robot Simulation and Environment

The first part of our framework involves the development of an accurate simulator to “stand in” for the actual robot. In prior work, we have developed models for the kinematics and inverse kinematics for this robot [25]. We use these to create a simulation of the robot, implemented in Simulink, in which the robot's motors are each represented as third-order transfer functions. The simulator is designed to be a modular component of the system, in the sense 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 2 (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

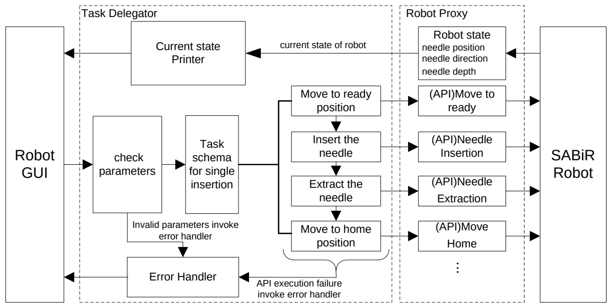

Fig. 3. Software Architecture.

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 [26].

We use a simple low level controller to control the simulation. 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 PD controller to follow this trajectory to guide the needle to the end point.

C. Software Architecture

We build a software system on top of the low-level controller. When a user interacts with the robot, this is what they see. This system has three components: a GUI, a task delegator, and a robot proxy. Figure 3 shows the information flow between these when the robot performs a high level insert/extract needle operation.

The software we build 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 command, the GUI then lists all parameters whose values are needed 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 then invokes the robot API to handle these decomposed 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 from the robot (or simulation). This design ensures that when the simulation is replaced by the actual robot, only the last stage needs to be modified. Specifically, additional communication and synchronization code will be used to handle communications with the robot's controller (built on XPC) and to ensure that data collection between the software and hardware is properly logged.

D. Data Collection

We build the entire architecture to be easy to monitor and log. We base the data collection subsystem on the producer/consumer synchronization technique. We collect software execution data at the function level. That is, calling each function in the robot API creates an object, which records the software state of the function's execution. This object is sent to a buffer queue through the producer. When the hardware data is available, the consumer thread starts to get the corresponding software state object from the queue and records both software data and hardware data in a user readable format. For example, when the robot begins to insert the needle, two software state objects are created and sent to the buffer queue. The first records the software states like action name "insert", depth of the needle inside tissue during the reference trajectory generation. The second records variables such as Motor Speed Error, which are estimated by the software after hardware data is available. Then another object is created to record only the hardware data of the action "insert" and sent to the buffer queue. When the consumer is not writing, it continuously takes objects out of the buffer queue and puts them into a local queue. When the consumer gets an object with only hardware data, it starts writing. While it is recording the data, the system can continuously transmit additional software data through the Producer as long as the buffer queue is not full.

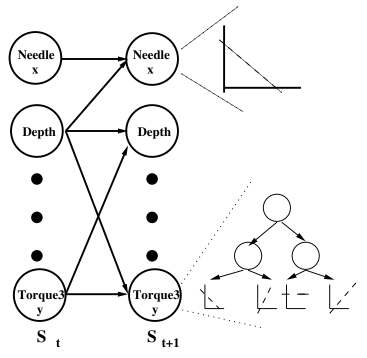

Fig. 4. Schematic of dynamic Bayesian network models we use showing linear Gaussian and regression tree CPDs.

E. Modeling the Hardware/Software State

We use the data we collect to model the time-evolution of the joint software/hardware state space of the system. To do this we use dynamic Bayesian networks (DBNs) [27]. These are first order Markov models that represent, in a factored manner, 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. We further consider structures that have no "symmetric" edges, that is, no edges from X_{t+1} to Y_{t+1}, where X and Y are state variables. Thus the factored state transition distribution according to the DBN is defined by:

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

where $\mathbf{V}_t = \{V_t^i\}$ denotes all of the variables in the t^{th} time step. A schematic DBN structure is shown in Figure 4. The 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 constant 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 [28] for the CPD. Each internal node of the regression tree is a test on some variable at the previous time step (Figure 4). Each leaf node is again a linear Gaussian model. Regression tree models such as these are a very general representation of nonlinear dynamics; effectively, each path through the tree partitions the input space into mutually disjoint

TABLE I

State Variables in DBNs.

Type	Name	Num	Description
Parameter	Reference Trajectory	18	Reference position, starting position, end position
	Ready Position	6	Position and orientation where insertion begins
	Insert Distance	1	Insert distance into tissue from ready position
Software	Action	2	High level action, e.g. "Insert Needle"
	Insert Depth	1	Depth of the needle inside tissue
	Needle Force	3	Force on the needle
	Needle Torque	3	Torque on the needle
	Motor Position Error	5	Error between Actual Position and Reference Position
	Motor Speed Error	5	Error between Actual Speed and Reference Speed
Hardware	Needle Position	6	Position and orientation of needle tip
	Motor Position	5	Positions of 5 motors
	Motor Torque	5	Torques of 5 motors

regions where a linear response is a good fit, and each partition can be as fine as required.

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." These variables are inputs to the system and do not change over time. While we use these parameters to help predict future state trajectories, they are not themselves predicted since they are constant. Other variables such as the x position of the needle tip are "hardware variables." The values for these variables are obtained by sensors on the robot or by direct hardware measurements of various sorts. The third type of variables are "software variables." These variables include flags denoting which high level motion is being executed, set by subroutines in the software. For example when the needle is being inserted into tissue, a flag is set indicating this high level action by the subroutine controlling this operation. Software variables also include 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. Such estimates can be used to estimate yet other variables such as "depth of needle in tissue." The set of variables describing the state is shown in Table I. For each kind of variable, "Num" refers to the number of variables of that kind, e.g. there are 3 needle forces, one in each direction.

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 simple

TABLE II

Average Test r^2 for normal state trajectories.

Model	Hardware	Software	All
HS	0.798	0.415	0.681
HS10	0.774	0.397	0.659
HWOnly	0.725	N/A	N/A

closed form solutions. For regression tree models, we use a standard greedy top down recursive decomposition approach [28]. At each point, each variable and its associated values are evaluated to estimate the benefit of splitting on that (variable, value) pair. All splits are binary. The benefit is computed using a weighted-average r^2 metric which compares the r^2 goodness-of-fit value before and after splitting (the weight is the fraction of points that go to each partition after a candidate split). The (variable, value) pair with the maximum gain is selected as the internal node. We make one modification to this standard tree construction procedure. In a normal case, the number of datapoints decreases as we go deeper into the tree because of the 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 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 decisions about the choice of splits. Further, to prevent overfitting, we prune the produced regression tree for each variable using a fixed validation set. We use a standard greedy post-pruning approach to do this [29].

IV. 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 2 center). The task for the robot is (following a reference trajectory) to insert the needle tip a specified distance within the tissue. We consider two kinds of A&A events. The first is an "encoder failure" event, where at some point within the trajectory, the element reporting a motor's position is lost, so the system can longer track that motor's position (Figure 2 right). The second is a "sweep" event, where prior to needle insertion, the needle tip strays and grazes the tissue surface. We generate several trajectories for each such event. Since actual A&A events are rare, we restrict

TABLE III

DETECTION AND PREDICTION TIMES (MS) AT FPR=0.06

Model	Detection		Prediction, Sweep	
	Encoder	Sweep	k=100	k=1000
HS	1	-	788	1532
HS10	10	10	720	1508
HWOnly	1	-	829	1585

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 I (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 then produce predicted states $\hat{s}_{t+1}$ and compute an r^2 metric that represents the accuracy of these predictions. These results are shown in Table II. From these results we observe that the DBNs are quite good at modeling the time-evolution of the hardware variables. It is more difficult to predict the software variables, though there is a lot of variability: some variables, such as the depth, can be very well predicted ($r^2 = 0.99$), and others such as the force on the needle in the y direction, are hard ($r^2 = 0.4$). However, as we show below, this still often allows us to classify, detect and predict A&A events. Finally, we observe that when using the HWOnly model, the accuracy of prediction on the hardware variables is less than when using the HS model. This suggests that the software variables add value to the DBN, and using them results in more accurate predictions.

Classifying trajectories. We next consider the question of classifying trajectories according to whether they contain an A&A event. This may be useful if execution data is logged on a remote system and later fetched for analysis. We consider a test set of 400 normal trajectories and 5 trajectories each of the two A&A events described above. For each point s_t on each trajectory, we use the DBN models to compute the negative log likelihood of s_{t+1} (or s_{t+10} for HS10). We record the maximum negative log likelihood (NLL) over the trajectory, and use this score to rank trajectories so that those with the highest NLL appear first. We know the ground truth for each trajectory, i.e. whether it contains an A&A event or not. We then create a receiver-operating characteristic (ROC) graph from this ranked list of predictions. ROC graphs show how the true positive rate (TPR) varies with the false positive rate (FPR) as a threshold is moved over a confidence measure (i.e. the deviation in our case). The result is shown in the first column of Figure 5.

From these results we see that a trajectory with an encoder failure is quite easy to detect for all our models. In fact HS and HWOnly are perfect at this task. This is probably because this event has a very strong signature among the hardware variables. The sweep event, on the other hand, is harder to detect if software variables are not used. This is likely because one of the variables in the HS model is the force on the needle tip, which behaves strongly abnormally in such cases and causes a large deviation from the expected behavior. Even though HWOnly is relatively less accurate, however, all of the DBNs are quite accurate in absolute terms and reach 100% TPR within 3% FPR.

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 again use a test set of 400 normal trajectories and 5 trajectories for each of the two 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."As before, we use the DBNs to check every (s_t, s_{t+1}), but in this case we do not aggregate the prediction over trajectories. Rather we associate each point with a smoothed 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 center column of Figure 5. 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 this threshold. These times-to-detect are shown in Table III (lower is better).

From these results we observe that while all the models are good at detecting encoder failures, the HS10 models have a small 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 can both accurately detect this event, while the HWOnly model lags behind. Again, this is probably because the sweep event is easier to detect with software variables than hardware variables alone. Finally, from Table III

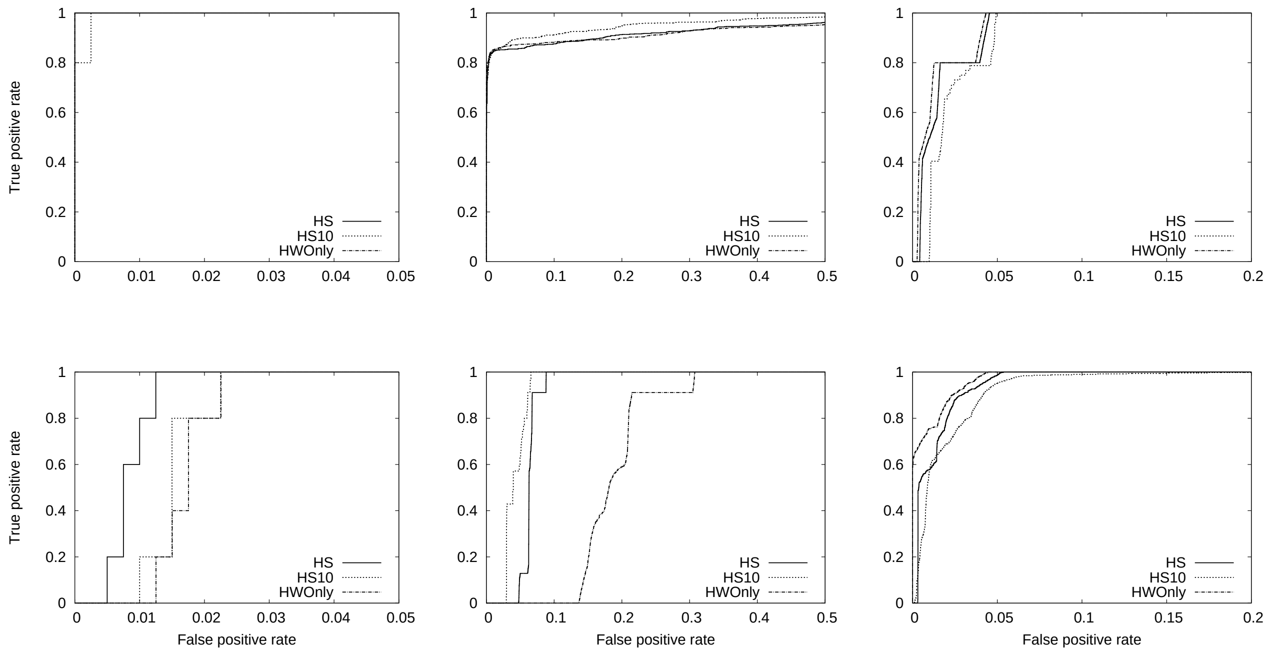

Fig. 5. ROC graphs for classification, detection and prediction of A&A events. The x-axes of some are truncated to better illustrate differences at low false positive rates. **Left column**: Classification, top:encoder failure; bottom, sweep. HS is best. **Middle column**: Detection, top: encoder failure; bottom, sweep. HS10 is best. **Right column**: Prediction, top: sweep (100 steps); bottom, sweep (1000 steps). HWOnly is best.

(left) we observe that all models can quickly detect encoder failures (within 1-10ms after it happens), 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 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 only the needle tip positions and measure the average NLL score of these points compared to the *reference* trajectory that is provided as input. (We use only the needle tip position because that is the only information in the reference trajectory.) In other words, 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 computed 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 last column of Figure 5, we show the results for $k = 100$ (0.1s, top) and $k = 1000$ (1s, bottom) for the sweep event. We also measure the time-to-event (Table III 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 HWOnly does very well, closely followed by HS. 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 is limited in this setting. From Table III, we see that the HWOnly model is also able to predict an A&A event earlier than the other models, though again, HS is close.

To summarize, all of the DBNs we test perform well in our experiments at different tasks. HS is the best at classifying trajectories, HS10 is best at detecting A&A events after the fact, and HWOnly is best at predicting A&A events before they happen. However, we also see that even where it is not the best, HS is a close second, indicating that this model is a good compromise for all three tasks that we consider. These results also indicate that DBNs of the type we use can be successful in identifying A&A events as we expand the framework.

V. Conclusion

We have proposed a framework to improve the reliability of medical robots. We have described a prototype that addresses some of the subproblems, and presented encouraging preliminary results that indicate the feasibility of building on this foundation.

A lot remains to be done. We are currently concurrently improving the simulation and controller by adding path planning and a simulation of image-guidance, improving the software architecture to enable more complex high level actions to be performed and collect more data about the software execution, and finally also adding to the statistical analysis by computing "usage envelopes," that represent regions of the state space where the robot can operate *reliably*, without many A&A events. We are also considering alternative A&A events to model. In the next stage, we plan to integrate with the hardware and evaluate how well we can detect A&A events there, as well as use the results to refine the software/simulation framework.

VI. Acknowledgments

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

References

[1] R. H. Taylor and D. Stoianovici, "Medical robotics in computer integrated surgery," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 5, pp. 765–781, 2003.

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

[3] FDA, "Adverse event report 2955842-2008-01144: Intuitive surgical inc., Da Vinci S Surgical System endoscopic instrument control system," www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfMAUDE/Detail.CFMMDRFOI__ID=1077464, July 2008.

[4] Intuitive Surgical Inc., "Da Vinci S Surgical System," www.intuitivesurgical.com/products/ davincissurgicalsystem/index.aspx, 2009.

[5] B. L. Davies, *Computer-Integrated Surgery: Technology and Clinical Applications*. MIT Press, 1996, ch. A Discussion of Safety Issues for Medical Robots, pp. 287–300, R. H. taylor et al., Editors.

[6] E. Dombre, P. Poignet, F. Pierrot, G. Duchemin, and L. Urbain, "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 Environment*, 2001.

[7] R. D. Howe and Y. Matsuoka, "Robotics for surgery," *Annual Review of Biomedical Engineering*, vol. 1, pp. 211–240, 1999.

[8] W. S. Ng and C. K. Tan, "On safety enhancements for medical robots," *Reliability Engineering System Safety*, vol. 54, no. 1, pp. 35–45, 1996.

[9] G. Duchemin, P. Poignet, E. Dombre, and F. Peirrot, "Medically safe and sound [human-friendly robot dependability]," *Robotics & Automation Magazine, IEEE*, vol. 11, no. 2, pp. 46–55, 2004.

[10] S. B. Ellenby, "Safety issues concerning medical robotics," in *Safety and Reliability of Complex Robotic Systems, IEE Colloquium on*. IET, 1994, pp. 3–1.

[11] A. Donzé, B. Krogh, and A. Rajhans, "Parameter synthesis for hybrid systems with an application to simulink models," in *Proceedings of the 12th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 165–179.

[12] R. Jetley, S. P. Iyer, and P. Jones, "A formal methods approach to medical device review," *Computer*, vol. 39, no. 4, pp. 61–67, 2006.

[13] B. Fei, W. S. Ng, S. Chauhan, and C. Kwoh, "The safety issues of medical robotics," *Reliability Engineering System Safety*, vol. 73, no. 2, pp. 183–192, 2001.

[14] Y. Hu, T. Podder, I. Buzurovic, K. Yan, W. Ng, and Y. Yu, "Hazard analysis of EUCLIDIAN: An image-guided robotic brachytherapy system," in *Proceedings of the 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS)*, 2007.

[15] P. Varley, "Techniques for development of safety-related software for surgical robots," *IEEE Transactions on Information Technology in Biomedicine*, vol. 3, no. 4, pp. 261–267, 1999.

[16] B. Halder and N. Sarkar, "Robust fault detection of a robotic manipulator," *International Journal of Robotics Research*, vol. 26, no. 3, pp. 273–285, 2007.

[17] R. Mattone and A. D. Luca, "Relaxed fault detection and isolation: An application to a nonlinear case study," *Automatica*, vol. 42, no. 1, pp. 109–116, 2006.

[18] M. L. McIntyre, W. E. Dixon, D. M. Dawson, and I. D. Walker, "Fault identification for robot manipulators," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 1028–1034, 2005.

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

[20] T. Mikaelian, B. Williams, and M. Sachenbacher, "Model-based monitoring and diagnosis of systems with software-extended behavior," in *Proceedings of the* 20^{th} *National Conference on Artificial Intelligence (AAAI)*, Pittsburgh, PA, 2005.

[21] R. R. Lutz and I. C. Mikulski, "Operational anomalies as a cause of safety-critical requirements evolution," *Journal of Systems and Software*, vol. 65, no. 2, pp. 155–161, 2003.

[22] J. A. Whittaker and M. G. Thomason, "A markov chain model for statistical software testing," *IEEE Transactions on Software Engineering*, vol. 20, no. 10, pp. 812–824, 1994.

[23] G. K. Baah, A. Podgurski, and M. J. Harrold, "Causal inference for statistical fault localization," in *Proceedings of the 19th international symposium on Software testing and analysis*. New York, NY, USA: ACM, 2010, pp. 73–84.

[24] O. Bebek, M. J. Hwang, B. Fei, and M. C. Çavuşoğlu, "Design of a small animal biopsy robot," in *Engineering in Medicine and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE*. IEEE, 2008, pp. 5601–5604.

[25] M. J. Hwang, O. Bebek, F. Liang, B. Fei, and M. C. Çavuşoğlu, "Kinematic calibration of a parallel robot for small animal biopsies," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, 2009, pp. 4104–4109.

[26] R. C. Jackson and M. C. Çavuşoğlu, "Modeling of needle-tissue interaction forces during surgical suturing," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), St. Paul, MN, USA*, May 14-18 2012, to appear.

[27] T. Dean and K. Kanazawa, "A model for reasoning about persistence and causation," *Computational Intelligence*, vol. 5, no. 3, pp. 142–150, 1990.

[28] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[29] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.