A Software Framework for Simulation of Safety Hazards in Robotic Surgical Systems

Homa Alemzadeh, Daniel Chen, Zbigniew Kalbarczyk, Ravishankar K. Iyer
Coordinated Science Laboratory, University of Illinois at Urbana-Champaign
{alemzad1, dchen8, kalbarcz, rkiyer}@illinois.edu

Xiao Li, Thenkurussi Kesavadas
Health Care Engineering Systems Center, University of Illinois at Urbana-Champaign
{xiaoli16, kesh}@illinois.edu

Jaishankar Raman
Division of Cardiac Surgery, Rush University
jai_raman@rush.edu

ABSTRACT
This paper presents a simulation framework for recreating the realistic safety hazard scenarios commonly observed in robotic surgical systems, which can be used to prepare surgical trainees for handling safety-critical events during procedures. The proposed simulation platform is composed of a surgical simulator based on an open-source surgical robot platform, Raven II, integrated with a software-based fault-injection engine, which automatically inserts faults into different modules of the robotic software. We demonstrate the value of software-based fault injection for simulating representative safety hazards seen in the adverse events reported to the FDA MAUDE database, by performing experiments both in simulation and on the actual Raven II robot.

General Terms
Measurement, Reliability, Experimentation, Human Factors.

Keywords
Safety, Simulation, Fault Injection, Training, Robotic Surgery.

1. INTRODUCTION
Use of robotic systems for minimally invasive surgery has increased significantly during the last decade. Between 2007 and 2013, over 1.74 million robotic procedures were performed in the U.S. across various specialties of gynecology, urology, general, cardiothoracic, and head and neck surgery [1]. Our comprehensive analysis, covering over 10,000 adverse robotic-surgery-related events during 2000–2013 as reported to the U.S. Food and Drug Administration (FDA) Manufacturer and User Facility Device Experience (MAUDE) database, showed that despite significant improvements in robotic technology through the years and broader adoption of the robotic approach, there are ongoing occurrences of safety incidents that negatively impact patients. The number of injury and death events per procedure has stayed relatively constant since 2007, with an average of 83.4 events per 100,000 procedures (with a 95% confidence interval of 74.2–92.7). In particular, out of a total of 10,624 events reported during 2000–2013, 1,535 (14.4%) had significant negative patient impact (including *injuries* (1,391 cases) and *deaths* (144 cases)) and 970 (9.1%) contributed to *procedure interruptions*, such as manual reset of the system, conversion of the procedure to non-robotic surgery, or rescheduling of the procedure to a later time [2].

The ability of current robotic surgical technology to automatically mitigate the impact of safety-related incidents is not comparable to the situation in other safety-critical industries, such as commercial aviation. In such industries, great effort has been spent over the years on improving safety practices by providing comprehensive simulation-based training that includes operation in the presence of safety-critical failures [3]. Multiple studies have shown that simulation can be effectively used in training to improve skill levels of robotic surgeons. There are already several surgical simulators, training centers, and validated curricula for robotic surgery [4]–[11]. However, the emphasis has been only on improving surgical skills and not on handling safety-critical events and responding to technical problems. Adverse events or machine failures are rarely used as potential scenarios for safety training of surgical teams.

Motivated by the idea of simulating safety hazards during robotic surgery training in order to prepare surgeons for handling safety-critical events, we created a platform to demonstrate the feasibility of using software-based fault injection to simulate realistic safety hazard scenarios with minimal change to the robotic software and hardware.

Software-implemented fault injection (SWIFI) [12] is commonly used for evaluating the safety and reliability of computing systems [13][14]. SWIFI validates the effectiveness of fault-tolerance mechanisms by studying the behavior of a system in the presence of faults. However, we use software-based fault-injection techniques to enable evaluation of human operator performance and response to safety hazards in simulation-based training.

Medical Cyber Physical Systems 2015, April 13, 2015, Seattle, WA

Table 1. Common types of device malfunctions and inadequate operational practices

Device and instrument malfunctions	Inadequate operational practices
- Master tool manipulator (mtm) malfunctions - Patient-side manipulator (psm) failures - Unintended operation of instruments (e.g., uncontrolled movements, power on/off) - Video/imaging problems at the surgeon's console - Recoverable and non-recoverable system errors - Burns and holes in tip cover accessories, leading to electrical arcing, sparking, or charring of instruments - Broken parts of instruments falling into patients	- Inadequate handling of emergency situations - Lack of training with specific system features - Inadequate troubleshooting of technical problems - Inadequate system/instrument checks before procedure - Incorrect port placements - Incorrect electro-cautery settings or cable connections - Inadequate manipulation of robot master controls - Inadequate hand and foot coordination by main surgeon - Incorrect manipulation or exchange of instruments

The proposed simulation platform was developed based on the Raven II robot, an open-source surgical robot developed by the Applied Dexterity and Biorobotics Lab at the University of Washington [15][16]. We integrated the robotic surgical simulator with a software-based fault-injection framework that simulates realistic safety scenarios observed in adverse event data during basic surgical tasks. We simulated the safety hazards caused by either device malfunctions or improper human operations by automatically injecting faults into the software modules of the robotic surgical simulator. Realistic safety hazards from adverse events reported to the MAUDE database were used to populate a library of hazard scenarios that define the location and type of faults and the conditions under which they should be injected into the RAVEN II software.

2. METHODOLOGY

This section presents our methodology, including the analysis of adverse event reports on robotic surgical systems from the MAUDE database, the surgical simulator based on RAVEN II, and the fault-injection framework for simulation of safety hazards extracted from data.

2.1 Analysis of Adverse Event Reports

Table 1 summarizes common types of device and instrument malfunctions and inadequate operational practices that contributed to catastrophic events or interruptions during robotic procedures.

During 2000–2013, a total of 9,382 (88.3%) of the reported events involved device and instrument malfunctions [2]. Table 2 shows samples of representative adverse events (from the MAUDE database) in which malfunctions of master tool manipulators (the inputs at the master console that the surgeon uses to send control commands to the robotic instruments) or motor encoders and potentiometers (the sensors at the patient side that collect measurements from robotic arms and instruments as feedback to the master console) or other device-related failures led the safety processor to stop the system and raise system errors during a procedure. The majority of system errors in Table 2 could not be resolved even by multiple system restarts and eventually led the surgical team to convert the procedure or abort and reschedule it to a later date. Between 2000 and 2013, out of 536 cases in which system errors were experienced during procedures, 91% led to procedure interruptions, including system resets (43%), procedure conversions (61.5%), and rescheduling (24.8%) [2]. (Note that those categories are not mutually exclusive. In some cases, after several system resets, the procedure was converted or rescheduled to a later date.)

Table 3 shows example events in which inadequate operator actions, due to deficiencies of the human-machine interface or lack of training, contributed to unexpected device operation and led to adverse impact on patients. None of the events in Table 3 resulted in a system error.

Although state-of-the-art robotic surgical systems are designed with safety mechanisms that detect failures and put the system into a recoverable or non-recoverable safe state, in practice those mechanisms are imperfect because of multiple factors.

Table 2. Example adverse event reports that involved device and instrument malfunctions

Report No. (Year)	Summary Description	Malfunction Type	Procedure Outcome
1006071 (2008)	- Recurring system errors #201 and #264, even after multiple restarts. - Errors due to voltage tracking faults and put the system in a recoverable safe state.	Master tool manipulators	Converted after 2 hours
3283230 (2013)	- Master tool manipulator arm was sluggish and could not control the robotic arms. - System error #22580 due to out-of-range hardware voltage level. - Multiple system restarts did not resolve the issue.		Aborted post anesthesia
3093014 (2013)	- Recurring error #23000, even after emergency power off & restart. - System error caused because the angular positions of one or more robotic joints on a manipulator as measured by the primary sensor (encoder) and secondary sensor (potentiometer) were out of range or in disagreement.	Joint sensors (Potentiometer or encoder)	Aborted post anesthesia and port incision
2916352 (2012)	- Recurring system error #23008, even after emergency power off & restart. - Recoverable errors caused because the angular positions of robotic joints as measured by the primary sensor (encoder) and secondary sensor (potentiometer) were out of range or in disagreement.		Converted after port incision
2014 (3620041)	- Non-recoverable error #23013 on patient side manipulator. - Multiple system restarts to recover from error but unsuccessful		Converted to open surgery

Table 3. Example adverse event reports that involved improper operational practices

Report No. (Year)	Summary Description	Inadequate Operational Practices	Procedure Outcome
921167 (2007)	- Patient-side manipulator dropped suddenly. - Scissors instrument bumped into uterus.	Surgeon removed his/her hands from master manipulators before removing his/her head from console viewer (keeping head in the console viewer keeps the robot engaged)	Pierced patient's uterus
1570678 (2009)	- Endoscopic camera manipulator difficult to move. - Master tool manipulator drifted when released.		Punctured patient's uterus
1961862 (2010)	- Instrument moved to guided tool change mode, moved slightly forward, and bumped into colon.		Injury to patient's colon
2644122 (2012)	- Uncontrolled movement of master manipulators.		Damaged abdominal wall
2636117 (2012)	- Limited range of motion and drift while master tool manipulators were used, even after system restart.		Aborted after 1.75 hours
2476271 (2012)	- Monopolar energy was released when bipolar instrument was used.	Improper connection of bipolar instrument to electrosurgical unit	Injury to patient's bowel
3024317 (2013)			
2494890 (2012)	- Arcing from bipolar instrument when cautery energy was not being applied.		Small burn on diaphragm

1) The diagnostic mechanisms are usually not comprehensive enough to identify the root causes of malfunctions and system errors during surgery. Thus, information on the type of system error (e.g., in an error condition of recoverable or non-recoverable) and corresponding troubleshooting procedures are incorrectly communicated to the surgical team. The root causes are often determined only after the fact, when further investigations by the field service engineers are performed and the failure scenarios are replicated. For example, an encoder or sensor malfunction may be reported as a recoverable system error during a procedure, when it is not recoverable and can only be fixed by replacing the component after the procedure (See report No. 3035720 [17].)

2) System operators, including surgeons, surgical assistants, and field service engineers, are often not well trained to correctly interpret reasons for observed system errors and to choose efficient troubleshooting actions to recover from emergency situations. For example, in one event, it was reported that the surgical team spent a significant amount of time troubleshooting a non-recoverable system error while the patient was under anesthesia for more than one hour. (See report No. 1743065 [18].)

Results from our analyses show the importance of designing advanced safety mechanisms for active monitoring of system components and operator actions to assist surgical teams in predicting and preventing critical events and performing effective troubleshooting procedures. Also, surgical training on the use of robotic systems must be improved to make troubleshooting and handling of adverse events a central part of the training experience.

2.2 RAVEN II Robotic Surgical Simulator

The RAVEN II robot is an open-source platform for research in tele-operative robotic surgery. Figure 1 depicts a typical configuration of a robotic tele-surgery system, composed of a master console, a communication channel, and a RAVEN II surgical robot, including software and hardware components. As shown in Figure 2, we developed a surgical simulator based on RAVEN II by modeling the operation of the master console and robotic arms and instruments in software in order to enable running of RAVEN software without the robotic hardware. Thus, we have the flexibility of either using real human input from a haptic device or replaying previously collected trajectories from real surgical tasks, to study test case scenarios during which our fault-injection framework recreates the safety hazards on the actual robot or in simulation.

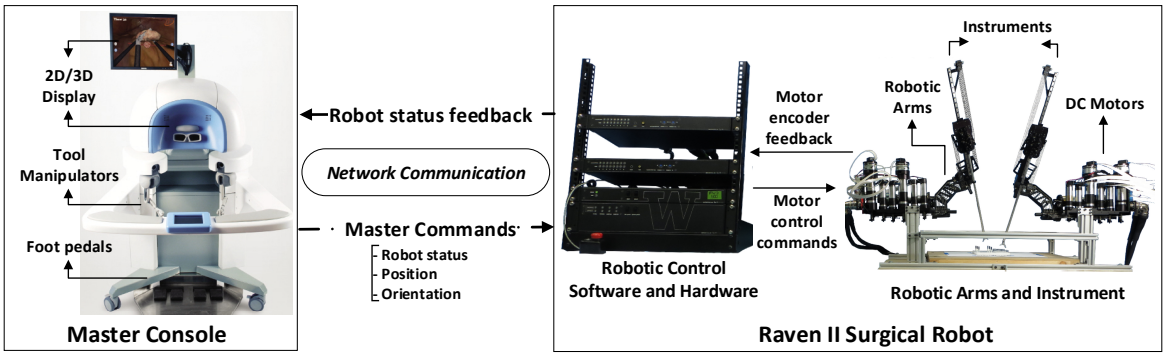

Figure 1. Robotic Tele-surgery using Raven II Surgical Platform (Modified from [19][20])

The **master console** provides the means for the surgeon to issue commands to the robot, using foot pedals and master tool manipulators. The desired position and orientation of robotic arms and robot control mode are transferred between the master and the slave robot over the network using the Interoperable Teleoperation Protocol (ITP), a protocol based on the UDP packet structure [21]. The RAVEN II software receives the master command packets and translates them into motor commands that enable the movement of robotic arms and instruments. We developed a software module that generates master command packets based on a previously collected trajectory of movements made by a human operator and sends them to the RAVEN control software.

The **RAVEN II software** runs on top of the Robotic Operating System (ROS) middleware and a real-time Linux kernel, communicating with the motor controllers through custom USB interface boards. A watchdog timer implemented on a PLC safety processor monitors the state of the RAVEN control software and moves the system into a fail-safe state upon detection of any problems. Three main threads run in parallel in the RAVEN control software: 1) a network-layer thread, which receives the command packets from the master controller over the network; 2) a control thread, in which the main robot kinematics and control computations are performed; and 3) the console thread, which provides an interface for setting robot control modes and displaying the robot's status to the user [20].

We simulated the **RAVEN II hardware** by developing a software module that mimics the dynamical behavior of the real robotic actuators by modeling the MAXON RE30 and RE40 DC motors used by RAVEN [15] as first-order systems with different time constants. A 3D virtual environment based on C++ OpenGL pipeline and CAD models of robot mechanical components was created to animate the movements of robotic arms and instruments.

Next, we integrated the RAVEN II surgical simulator with a fault-injection framework to simulate representative safety hazards observed in the MAUDE data.

2.3 Fault-injection Framework

We developed the fault-injection framework by using a combination of *compile-time* and *run-time* software fault-injection techniques. We recreated each hazard scenario by injecting faults into different parts of the simulator to emulate representative device malfunctions and improper human operations. The safety hazard scenarios are read from a hazard scenario library that specifies fault-injection parameters such as the *location* in the software, the *trigger* or condition under which the fault should be injected, and *target* variables to be affected by the injection. The faults are injected by replacing the code at the specified location with a mutated version that mimics the intended faulty operation (compile-time fault injection) [12] or by setting breakpoints at the specified location and changing the target variables at run-time (run-time fault injection) [12]. The fault-injection controller then collects the output from simulator modules (e.g., user input packets, output joint positions, and error messages), and the collected injection logs are sent with the usual surgical simulator logs to an online database for further analysis.

Run-time fault injection allows us to perform fault injection in run-time generated data; however, the delay introduced by the run-time fault injector is not suitable for modules that have hard real-time requirements (e.g., the kinematics and control computations). *Compile-time fault injection* requires modification and pre-compiling of the fault injection condition into the source code, but has negligible timing overhead, which makes it suitable for modules with real-time requirements. For example, to mimic the safety hazards due to malfunctions in the master tool manipulators, which send run-time packets to the RAVEN network-layer thread, we corrupt the packets received in the RAVEN network-layer thread through run-time fault-injection using a fault injector that remotely attaches to the RAVEN process in ROS. To mimic the safety hazards due

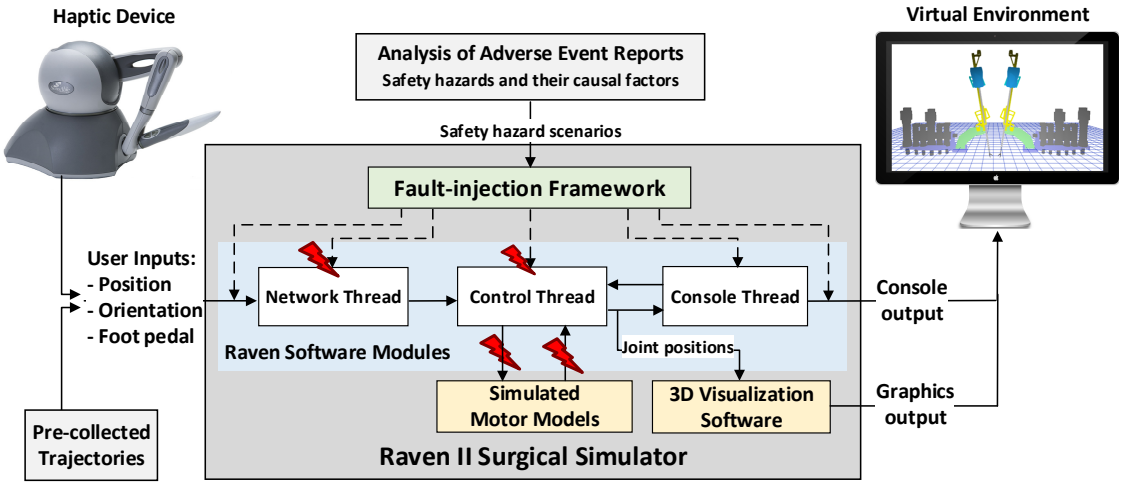

Figure 2. Raven II Surgical Simulator Integrated with Fault-injection Framework

to sensor (encoder) malfunctions and unintended instrument operations, we inject compile-time faults into the USB packet communication in the control thread. The RAVEN control thread has the hard real-time requirement of one millisecond to perform kinematics calculations and communication with the USB interface boards [20]. Run-time fault-injection to the control thread introduces small delays, leading to violation of the real-time constraint and failure of kinematics calculations, resulting in unintended robotic instrument vibrations and movements.

3. EXPERIMENTAL RESULTS

In preliminary experiments, we injected 5,500 faults into the network and control threads on the surgical simulator and 110 faults on the actual RAVEN II robot, while running a pre-collected trajectory of a simple surgical movement. Our purpose was to show that we can regenerate realistic safety hazard scenarios seen in the adverse events reported in the MAUDE database.

The following fault-injection cases describe simulation of the safety hazards shown in Tables 2 and 3 in the same way they were reported in the MAUDE data:

1. **System errors due to master tool manipulator malfunctions:** These malfunctions were simulated by injecting random faults into the ITP packets received by the network-layer thread. The fault-injection framework targeted the position, orientation, grasper angle, and foot pedal variables in the packet data structure by modifying their values into random values outside the range of possible values that the variables take. Since the ITP packets are sent in an incremental motion scheme, modifying the variables of a few packets to values inside the range of possible values did not have any impact, but intermittent injection of out-of-range values to the position, orientation, and grasper angle variables for an extended period causes the kinematics calculations to fail. The RAVEN II control software detects such failures by observing an over-the-limit electrical current command being sent from the software to the digital to analog converters (DAC) on the motor controllers and raises an E-STOP software error, leading the hardware watchdog timer to move the RAVEN hardware to an E-STOP safe state. The E-STOP error can only be resolved by restarting the system. Depending on the length of the faulty packets (e.g., if the master manipulator malfunction is permanent), the E-STOP error cannot be recovered from, even with multiple restarts.
2. **Sensor (encoder) malfunctions:** We simulated permanent and intermittent sensor malfunctions by injecting faults into different parts of the USB interface function responsible for communicating packets from motor controllers to the control thread through USB interface boards (*get_USB_packet*). Corruption of the number of available USB boards, indices for accessing USB boards, and packets read from the USB boards caused the RAVEN watchdog processor to detect an error and put the system in a non-recoverable E-STOP safe state, from which one cannot recover by pressing the restart button. Only a complete restart of the software and robotic hardware would resolve the issue.
3. **Improper human operation or patient-side manipulator malfunctions:** To simulate unintended instrument movements and sudden jumps of robotic arms due to device malfunctions or improper human operations, we injected faults into the motor command variables in the control thread and USB interface function (*put_USB_packet*). The injections into the USB packets sent from the control thread to the USB interface board caused abrupt jumps of the robotic arms, leading the RAVEN software and hardware to stop. Figure 3 shows the visualization of this safety hazard scenario in the 3D virtual environment.

Table 4 summarizes the above fault-injection cases in the actual robot. The last column shows the total number of injected faults and the number of faults that were manifested and caused the desired safety hazard scenarios. Faults injected into the network-layer thread were not manifested if they had been injected into the foot pedal variable or if faulty values were within the range of possible values (22/30 of faults injected into the network-layer thread were manifested.)

In the control thread, the faults injected into the USB board indices and packets in the *get_USB_packet* and *put_USB_packet* functions were not manifested as safety hazards during the robot initialization phase (homing) (61/64 and 13/16 of faults injected into *get_USB_packet* and *put_USB_packet* functions after the homing phase were manifested, respectively.)

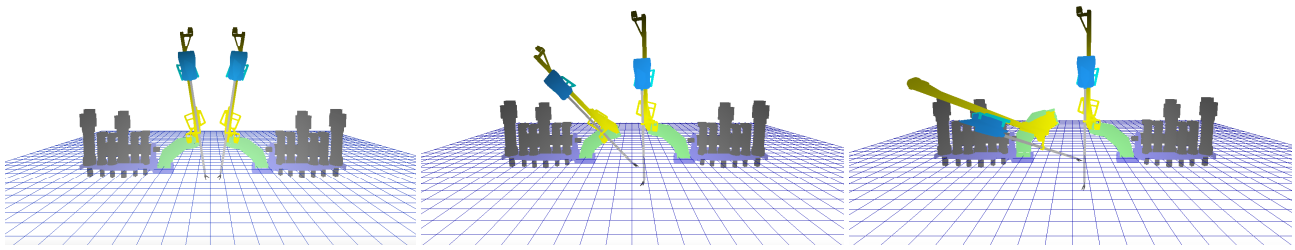

Figure 3. Visualization of a Safety Hazard Scenario in the Virtual Environment: The left robotic arm makes a sudden jump because of a faulty packet sent from control software to the motors

Table 4. Summary of fault-injection experiments on actual robot

Safety Hazard Scenario	Fault Type	Target Software Module	Injection Type	Target Variables	No. Manifested/ Injected Faults
Recoverable System Errors	Intermittent master tool manipulator malfunction	Network-Layer Thread	Run-time	Position, Orientation, Grasper angle, Foot pedal	22/30
Non-recoverable System Errors	Sensor (encoder) malfunctions	Control Thread *(get_USB_packet)*	Compile-time	USBboardsAt, Board index, Ret Value	61/64
	Improper human operation or patient-side manipulator malfunction	Control Thread *(put_USB_packet)*		USBboardsAt, Board index, Ret Value	10/12
Unintended Instrument Movements (sudden jumps)				Robotic joint commands	3/4*

* Since abrupt jumps of robotic arms could potentially lead to system damage, we repeated these injections only four times.

4. CONCLUSIONS

We developed a robotic surgical simulator augmented with fault-injection capabilities to show the feasibility of simulating safety hazard scenarios, which are commonly reported in the use of robotic surgical systems. The proposed framework for safety-hazard simulation can be used in designing future surgical simulators for safety training of surgeons to prepare them for handling common types of adverse events experienced during procedures.

The proposed fault-injection framework further provides the means for safety-based design of next-generation robotic surgical systems by evaluating the robustness of safety and fault-tolerance mechanisms with respect to realistic safety hazards previously reported in the field.

5. ACKNOWLEDGMENTS

A non-restricted grant from Infosys and a faculty award from IBM partially supported this work. Our special thanks to Andrew Lewis, David Drajeske, and Blake Hannaford from Applied Dexterity and researchers at the University of Washington Biorobotics Lab for providing the open-source code for the RAVEN II surgical robot and allowing us to perform fault-injection experiments on a RAVEN robot in their lab. We also thank Jenny Applequist for her editing of the paper.

6. REFERENCES

[1] Annual Report 2013, Intuitive Surgical, Inc.: http://phx.corporate-ir.net/External.File?item=UGFyZW50SUQ9MjIzOTk3fENoaWxkSUQ9LTF8VHlwZT0z&t=1

[2] H. Alemzadeh, J. Raman, N. Leveson, and R. K. Iyer, "Safety Implications of Robotic Surgery: A Study of 13 Years of FDA Data on da Vinci Surgical Systems." University of Illinois Coordinated Science Laboratory Technical Report, UILU-ENG-13-2208 (2013), In *Proc. of the 50th Annual Meeting of the Society of Thoracic Surgeons (STS)* (Jan. 2014), Orlando, FL.

[3] F. Bilotta, et al. Impact and Implementation of Simulation-based Training for Safety. *The Scientific World Journal* (2013).

[4] R. Smith, et al. Comparative Analysis of the Functionality of Simulators of the da Vinci Surgical Robot. *Surgical Endoscopy* (2014), 1-12.

[5] J. Bric, et al. Proficiency Training on a Virtual Reality Robotic Surgical Skills Curriculum. *Surgical Endoscopy* (2014), 1-6.

[6] A. Huser, et al. Simulated Life-Threatening Emergency During Robot-Assisted Surgery." *Journal of Endourology* (2014).

[7] K. Foell, et al. Robotic Surgery Basic Skills Training: Evaluation of a Pilot Multidisciplinary Simulation-based Curriculum. *Canadian Urological Association Journal*. 7, 11-12 (2013), 430-434.

[8] A. Patel, et al. Can We Become Better Robot Surgeons through Simulator Practice? *Surgical Endoscopy*. 28, 3 (2014), 847-53.

[9] S. Seixas-Mikelus, et al., Face Validation of Novel Robotic Surgical Simulator. *Urology (Gold)*. 76, 2 (2010), 357-360.

[10] A. P. Stegemann, et al. Fundamental Skills of Robotic Surgery: A Multi-institutional Randomized Controlled Trial for Validation of a Simulation-based Curriculum. *Urology.* 81, 4 (2013), 767-774.

[11] A. Chowriappa, et al., Development and Validation of a Composite Scoring System for Robot-assisted Surgical Training: The Robotic Skills Assessment Score. *Journal of Surgical Research.* 185, 2 (2013), 561-569.

[12] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer. Fault Injection Techniques and Tools. *Computer.* 30, 4 (1997), 75-82.

[13] D. Chen et al. Error Behavior Comparison of Multiple Computing Systems: A Case Study Using Linux on Pentium, Solaris on SPARC, and AIX on POWER. In *Proc. 14th IEEE Pacific Rim Intl. Symposium on Dependable Computing* (Dec. 2008)*, PRDC*.

[14] W. Gu et al. Characterization of Linux Kernel Behavior under Errors. In *Proc. International Conference on Dependable Systems and Networks* (June 22-25, 2003), 459-68.

[15] B. Hannaford, et al. RAVEN-II: An Open Platform for Surgical Robotics Research. *IEEE Transactions on Biomedical Engineering.* 10, 10 (2012).

[16] "RAVEN II Open-source Surgical Robots." ROS.org News: http://www.ros.org/news/2012/01/raven-ii-open-source-surgical-robots.html.

[17] MAUDE Adverse Event Report: Intuitive Surgical, Inc. da Vinci Surgical System Endoscopic Instrument Control System, MDR report 3035720, Mar. 5, 2013, U.S. Food and Drug Administration**:** http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfmaude/Detail.CFM?MDRFOI__ID=3035720.

[18] MAUDE Adverse Event Report: Intuitive Surgical, Inc. da Vinci Surgical System Endoscopic Instrument Control System, MDR report 1743065, May 26, 2010, U.S. Food and Drug Administration: http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfmaude/Detail.CFM?MDRFOI__ID=1743065.

[19] Robotic Surgery Simulator (RoSS), Simulated Surgical Systems: http://www.simulatedsurgicals.com/.

[20] "RAVEN II Source Code," University of Washington, http://brl.ee.washington.edu/raven2docs/.

[21] H. H. King, et al., Plugfest 2009: Global Interoperability in Telerobotics and Telemedicine. In *Proc. Int. Conf. Robot. Autom.* (May 2010), *ICRA 2010*, 1733–1738.