2012 12th International Conference on Control, Automation and Systems
Oct. 17-21, 2012 in ICC, Jeju Island, Korea

A n-dimensional Convex Hull Approach for Fault Detection and Mitigation for High Degree of Freedom Robots Humanoid Robots

Kevin Lynch[1], Daniel M. Lofaro[2] and Paul Oh[3]

[1]Department of Computer Science,
[2]Department of Electrical and Computer Engineering,
[3]Department of Mechanical Engineering,
Drexel University, Philadelphia, PA, USA
kml43@cs.drexel.edu, dml46@drexel.edu, paul@coe.drexel.edu

Abstract: This work shows a plan for error state determination, diagnosis, and mitigation using autonomic computing tools and tecniques on the Hubo robot. An n-dimensional geometric enclosure is constructed from periodic measurements of the robot's normal operating state. Similar hulls will be constructed for unique faults encountered during runtime. A mapping of these faults to applicable mitigations will be dynamically constructed and will aid in mitigation selection. The successful application of the mitigation will bring the robot back to a safe operating state.

Keywords: Humanoid Robotics, Fault Detection, Error Mitigation, n-dimensional Convex Hull

1. INTRODUCTION

April 27th, 2010 - Philadelphia Convention Center (Main Hall): The robot handlers Daniel M. Lofaro and Robert Ellenberg were preparing the adult-size humanoid robot Jaemi Hubo[1] at an outreach demonstration for the *Arts & Science Council* of Philadelphia. During the dress rehearsal one of Jaemi's actuators failed while preforming an active balancing demonstration. This resulted in Jaemi Hubo falling off the 4 foot high stage. The results of the impact can be found in Fig. 1 and on YouTube[2]. This annual event was a high profile fund raiser for the arts and science programs throughout the greater Philadelphia area and was covered widely by the media. If this failure would have occurred during the actual event the aftermath would have been even more devastating. Jaemi Hubo was repaired in full by Lofaro, Ellenberg and the rest of the Drexel Autonomous Systems Lab (DASL) within 50 days of the accident, see Fig. 2. However in order for outreach events like this to successfully continue, methods for detecting failure states and quickly choosing appropriate mitigations must be developed.

All electro-mechanical systems have an inherent mean-time to failure. Even with good maintenance these systems can fail without warning. This work proposes ways to *detect* when entering a failure state and ways of *mitigating* such failures. A failure state is defined as the operating conditions where the robot is unable to safely preform tasks. This includes, but is not limited to, actuator faults and failures, loss of balance, power loss, etc. Faults are defined as the failure (intermittent or perpetual) in a single part of the robot. A single fault does not mean a system failure in all cases. The adult-size humanoid robots Hubo2+ and Jaemi Hubo (KHR-4) are the primary test platforms for this proposed work. All methods used are written in a broad scope so it can be applicable to other electro-mechanical systems.

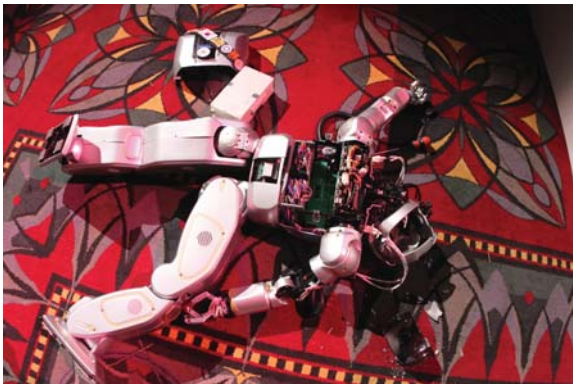

Fig. 1 - Aftermath of the 4 foot fall Jaemi Hubo took after one of her actuators failed during operation. A video with more images of the aftermath of the failure and further explanation of the event can be seen on YouTube[1].

Faults are difficult to detect before an executing system reaches a point of failure, as the first symptom of a fault is often system failure itself. While it is unrealistic to expect complex systems to be fault-free, actions such as resetting the system, quarantining specific components, or minimizing damage from the fault can be taken. Autonomic systems, an extension of fault tolerant systems, attempt to detect, diagnose, and mitigate faults quickly. These systems are inspired by the autonomic nervous system in the human body that monitors and regulates vital functions of the body such as heart rate, respiration rate, and digestion. Similarly, an autonomic computer system is able to monitor itself and its environment and automatically adapt to

*This project was supported by the Drexel Autonomous Systems Lab (DASL) and by a National Science Foundation - Partnerships for International Research and Education grant (#0730206).
*Hubo was designed and created by our partner Dr. Jun-Ho Oh, Department of Mechanical Engineering, Korean Advanced Institute of Science and Technology, Daejeon, South Korea.
[1]Jaemi Hubo Home Page: http://dasl.mem.drexel.edu/HUBO
[2]Jaemi Hubo Fall: http://www.youtube.com/watch?v=DF8zAM4FLB4

978-89-93215-04-5 95560/12/$15

complex changes. The goal of autonomic computing is to specify the desired state of a system using high-level objectives without detailing how to arrive at the state [1], [2], [3]. By making intelligent decisions, autonomic systems free system administrators from low-level management and the intricacies of complex systems. Autonomic systems aim to be self-configuring, self-optimizing, self-healing, and self-protecting. These properties, collectively referred to as the self-* properties, are different views of the same self-management property. For instance, a self-protecting system is ideally healing itself from faults while optimizing and reconfiguring itself to prevent other faults from reoccurring.

In this work Section 3.2 outlines the authors plan to use their bleeding-edge software engineering failure state detection techniques on the complex electro-mechanical system Jaemi Hubo. Section 3.3 describes the system faults used to determine failure states and how these faults are injected into the system in a controlled environment. Finally Section 3.4 shows how to define proper mitigation techniques for a give failure state and Section 4. describes our expected results.

Fig. 2 Jaemi Hubo 50 days after the fall at the Philadelphia Convention Center. Jaemi Hubo is once again in full operational order. She was fixed solely by the students at the Drexel Autonomous Systems Lab (DASL). This demonstrates the successful transfer of tribal knowledge of the Hubo platform from the Hubo Lab at KAIST to the Drexel Autonomous Systems Lab at Drexel University.

2. BACKGROUND

The technique proposed in this work borrows from the bleeding edge work in autonomic software engineering and computing. The heart of autonomic computing is anomaly detection, diagnosis, and mitigation. Autonomic systems perform 4 general tasks in a continuous

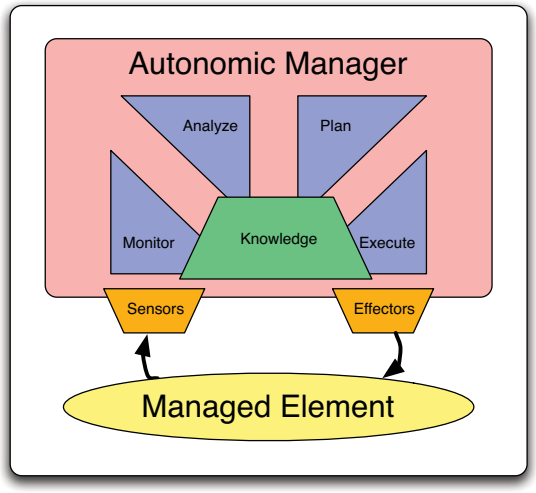

Fig. 3 - An autonomic element uses the MAPE control loop to monitor a component, analyze the its status with respect to a policy, plan how to meet the policy requirements, and execute a set of actions to do so.

closed loop: monitor components with the help of sensors, interpret the monitored data, create a repair plan for system adaptation, and execute this plan through effectors on the monitored system and its environment. This is the MAPE (Monitor, Analyze, Plan, Execute) loop [1]. Fig. 3 illustrates how an autonomic element uses this loop to manage a component.

Autonomic approaches can be applied to many different types of systems, particularly when focusing on fault detection and mitigation. self-healing opearting systems [4], [5] protect against bit-flips and other transient hardware faults, and are able to hot-swap components or firmware during runtime. At the application level, it is possible to monitor the architecture requirements of a system and detect when an architectural property (e.g. required connection bandwidth) falls outside of an acceptable threshold [6]. If an invariant is violated, it executes a strategy specifically designed to correct the violated invariant. The strategies can reinitialize components and reconfigure the system to recover from the violation and attempt to prevent the violation from reoccurring. However, the applied strategies are *ad hoc* solutions, requiring developers and maintainers to understand and preempt the shortcomings of the system, in much the same way they would manually debug and repair the system.

2.1 Fault Detection and Diagnosis

Previous approaches to the detection of software faults fall into two categories, signature-based and anomaly-based [7]. Signature-based methods detect faults by matching measurements to known fault signatures. These techniques are used in static fault-checking software such as the commercial antivirus software McAfee [8] and Symantec [9], as well as network intrusion detection systems such as Snort [10] and Netstat [11]. These

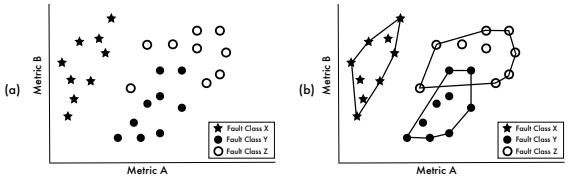

Fig. 4 - Building convex hulls for the diagnosis of faults (a) sample points from normal operation and two fault classes (b) Points inside of the convex hulls are diagnosed as normal or one of the fault classes. Points outside of all hulls are diagnosed as unknown faults.

techniques can also be used to detect recurring runtime faults [12].

Typically, anomaly-detection techniques begin by collecting sensor measurements of a normally behaving system. Then, they construct a representation of the monitored system and compare any future measurements against that representation. A common approach is to use metric correlations to quantify a monitored system. During detection, if the correlations between metrics becomes significantly different from the learned correlations, the system is classified to be in a faulty state [13], [14], [15], [16].

Once a fault is detected, it must be correctly diagnosed. Bayesian Network classifiers have been applied in several previous approaches to diagnosis using. Ghanbari et al. propose an approach to anomaly diagnosis based on Bayesian networks that are wholly or partially specified by a human user [17]. Tree augmented naive bayesian classifiers are the basis for software failure diagnosis in work by Cohen et al. [15]. Zhang et al. use ensembles of tree augmented naive Bayesian classifiers to diagnose faults. [18]. Other classification techniques have been used to diagnose software faults. Chen et al. offer an approach to the diagnosis of failures in large-scale Internet service systems using decision trees [19]. Duan et al. [20] propose an approach to diagnosis of software failures that uses active learning to minimize the number of data points that a human administrator must label.

In previous work it has been demonstrated that computational geometry can be used to effectively detect system faults at runtime[21], [22]. The approach, introduced in the work as Aniketos, is split into a training phase, and a monitoring phase. During the training phase, the monitored system executes its normal behavior and runtime data is periodically collected from sensors monitoring the application, the operating system, and the hardware. Using these measurements, we can construct an n-dimensional convex hull whose enclosing space represents the normal execution of the monitored application, where n is the number of distinct metrics used. Figure 4 shows an example of a 2-dimensional convex hull.

During the monitoring phase, if a measurement point falls outside of this enclosure, a fault is likely to have occurred. To properly mitigate the fault, the problem must be diagnosed [23] and a viable mitigation selected. If measurement data is available for a set of known faults, then it is possible to construct hulls for individual faults, or classes of faults. During the monitoring phase, if a measurement falls within one of these fault hulls we can determine which fault is likely occurring and apply a mitigation.This approach is also capable of recognizing that an occurring fault is not a type of fault that it has been trained to recognize. Typical classification techniques, such as naive Bayes and voting feature intervals[24], require that every sample be assigned a class from a finite list of possible classes, which can cause drastic consequences if an incorrect mitigation is selected. Our approach allows Aniketos to recognize that a fault has occurred, but does not force Aniketos to label the fault as one of the faults that it has been trained to recognize. One of the most important features of a fault detection and diagnosis system is the ability to handle new faults.

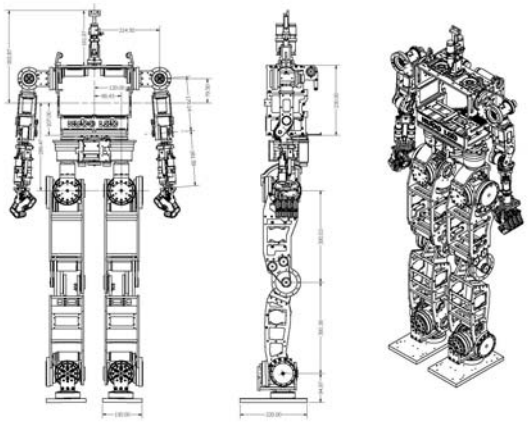

Fig. 5 Jaemi Hubo: 130cm tall 45kg (with battery and protective shell) 40 degree of freedom, high gain position controlled adult-size humanoid robot

2.2 Fault Mitigation

Once a fault is detected and diagnosed a mitigation can be applied in an attempt to bring the system back to a normal operating state. To select the most appropriate mitigation to execute, mitigations must be evaluated based on their performance cost, impact on the system, reliability, and how they affect other mitigations locally as well as in the global environment. Given a set of faults and a set of mitigations, it is possible to intuitively and empirically determine the best possible mitigation for a given fault. However, it is impossible to plan for all possible fault scenarios, limiting the effectiveness of mitigation selection.

Generally, when a fault is encountered there are several approaches that can be taken. Reinitialization brings the whole system, or part of the system, to its initial state when a fault is detected. If a fault can be localized, it is possible to progressively microreboot larger portions of the system until the problem is resolved, starting with the most localized component [25]. A generalization of reinitialization, known as rollback, restores the system to a previous checkpoint, undoing any environment changes

that may have been made [26]. While this approach may not always be feasible, in some cases, simply undoing the failing operation may be sufficient. Often times, the system needs to be reconfigured to adapt to changes in the system's environment. This can be done by tuning component parameters, scaling components, replacing components, or removing components entirely. Finally, in certain cases, it may be best to just accept the fault and continue without mitigation.

It is important to understand that each of these approaches could have potentially disastrous consequences on a given system. Different systems have different objectives and costs risks associated with it. Similarly, each mitigation has execution costs and risks associated with it. It is possible for a selected mitigation to cause more faults, or increased downtime. If an incorrect mitigation is applied, a cascading effect could quickly emerge.

2.3 Humanoid Robot Fault Mitigation

A typical fault for articulated robots is actuator failure. Upon failure there is no more power applied to the joint and the entire limb becomes under-actuated. This is particularly hazardous to robots that require feedback to balance such as biped humanoid robots. For biped humanoids these errors typically caused actuator over torque or hardware error resulting in loss of zero-moment-point (ZMP) [27] causing a robot fall or collapse. This is exceptionally harmful to adult size humanoid robots due to their weight. A common mitigation method described by Shin, J. et al.[28] from the Korean Advanced Institute of Science and Technology (KAIST) changes the model of the affected manipulator to one that is under-actuated. This new model allows the robust controller to continue to operate collision free (safe).

Current methods of mitigation of ZMP loss for biped humanoids been investigated by Kiyoshi Fujiwara et al. [29]. These methods involve finding an optimal falling trajectory that reduces the instantaneous force of the robot at impact by creating multiple impact stages[30]. This method was fully tested on an HRP-2FX (HRP-2P surrogate) and partially on an HRP-2P. This work did not include a method of determining a falling state, it is assumed that a fall is in progress. Additional work on detecting a fall and reducing fall damage has been shown by Kunihiro Ogata et al.[31]. An active shock-reducing motion reduces the impact damage by following the center of gravity (COG) and attempting to keep it close to the ZMP support polygon. The falling state is determined when the predicted ZMP departs from the support polygon. This method was tested on a miniature humanoid robot. Additional work on determining a fall state using machine learning techniques[32]. Reimund Renner et al. used parameter estimation of multiple sensors to detect a falling state[33].

3. METHODOLOGY

Our proposed process is divided up into two distinct parts, failure state determination (Section3.2) and mitigation (Section3.4). The determination techniques are designed to be wide in scope. The mitigation techniques are tailored towards adult-size humanoid robots, specifically the Hubo platforms described in Section 3.1.

3.1 Platform

The robots that will be used in these experiments are the adult-size humanoid robots Jaemi Hubo and Hubo2+, see Fig. 5. Each Hubo contains 40 degrees of freedom (DOF) and stands at a height of 130cm. Each actuator has a high gain internal PID position control loop. Reference positions are commanded at 100hz over a half-duplex 1.0Mbps controlled area network (CAN). Each actuator can feed back actual position, current through the actuator, and the actuator status (enabled or disabled). Other metrics that are fed back are described in Table 1 and Table 2.

Currently we have direct access to seven Hubo's in total, six Hubo2+ and one Hubo2 (Jaemi Hubo). Two of these robots will be used in these experiments for initial testing. Access to this common platform (Hubo) allows us to verify results with minimal mechatronic differences. This is the driving goal behind the NSF-MIR2 (Grant Number CNS-0960061) through which the Hubo platforms were procured.

3.2 Failure State Determination

To accurately detect when a fault is occurring in the Hubo robots, the Aniketos system requires an understanding of each robot's normal state. This normal state is constructed from a variety of safe routines that exercise the different features of the robots individually and simultaneously. If a fault is triggered during training, then any subsequent time the fault manifests, it will be presumed to be safe. If these routines do not exercise all features of the robots, then it is possible that a mitigation will be triggered when no fault is occurring. To construct the state, Aniketos collects data from all available sensors, treating each set of measurements as a point in n-dimensional space. Aniketos uses an online algorithm to construct the enclosing n-dimensional convex hull around these points as each each data point arrives.

During the monitoring phase, a new measurement point x is a member of the convex set $X = \{x_1, \ldots, x_k\}$ if

$$x = \sum_{i=1}^{k} \lambda_i x_i \tag{1}$$

In order to satisfy the convex hull criteria

$$\lambda_i \geq 0 \tag{2}$$

and

$$\sum_{i=1}^{k} \lambda_i = 1 \tag{3}$$

Table 1 - Actuator metrics analyzed for convex hull creation

Metric	Type	Status/Measurement
Over Current	Actuator	Status flag - motor controller has been shutdown due to over current
Status	Actuator	Status flag - motor controller's internal PID control loop is activated
Zeroed	Actuator	Status flag - motor controller has been zeroed
Responding	Actuator	Response - Responds if the motor controller is communicating properly
Commanded Position	Actuator	Reference position given to motor controller.
Measured Position	Actuator	Actual position of motor
Position Error	Actuator	Position error between commanded and measured position of the actuator
Current	Actuator	Current used by the given actuator

This can be solved using a nonnegative linear least squares algorithm[34]. If a feasible solution exists, then the point is contained in the convex set. The time needed to classify a single measurement point is dependent on the number of dimensions and the number of points needed to define the hull.

Aniketos determines that a fault is occurring when the current system state is outside of the normal state. To diagnose that a specific is occurring fault hulls must be created for each known possible fault. This fault data is collected in a fashion similar to the normal state, except the fault must be triggered in a controlled environment. Section 3.3 describes the faults that will be studied and how these faults are injected into the system in a lab setting.

Once all of the hulls are constructed, the monitoring phase will analyze each measurement point and determine whether the robot is operating normally. If it is not, Aniketos will attempt to determine if a known fault is occurring, or if an unknown fault is occurring. Typically, it takes less than 100ms to classify a point using 25 dimensions and a hull that is constructed from 10000 points using a single threaded 2.4Ghz x86 computer. To better meet the real-time data rate, algorithms may be used to reduce the number of points, and more efficient algorithm implements can be used to reduce the processing time. If a fault is detected, an appropriate mitigation should be selected and applied.

3.3 Fault Injection

Faults will be injected to the Hubo system in a lab setting. Each fault will be injected on eight separate trials to ensure statistically significant results. During each test system metrics are be recorded at a rate of 100hz for the entirety of the test. Table 1 and Table 2 describes each of the metrics recorded. Each test will consist of two stages. The first stage lasts 30 seconds. In this stage the robot is standing or walking (depending on the test) in a stable fault and failure free state. The given fault is injected at 30 seconds. This marks the beginning of the second stage. The second stage will last two minutes or 30 seconds after instability has occurred, which ever comes first.

The faults that will be injected are *actuator failure* and *ZMP loss*. Actuator failures will be created by:

- Removing power to actuator during operation (Actuator will no longer respond)
- Removing power to motor during operation (Actuator will respond but the motor has no power)
- Lowering the over current threshold within the actuator causing an over current shutdown.

ZMP loss will be tested by:

- Known impulse (push) applied to shoulder in the x and y directions.
- Step onto un-even terrain.
- Sudden ground shift in x and y direction (treadmill turned on then off)

To ensure no physical harm to the robot during testing, All test will be preformed in the full-scale safe testing environment designed for experiments with Jaemi Hubo created using DASLs Systems Integrated Sensor Test Rig (SISTR)[35].

A n-dimensional convex hull for the normal operating state and each of the injected fault states will be created using the above data using the methods described in Section 3.2.

3.4 Mitigation Analysis

The effectiveness of different mitigations has been analyzed on faults injected into two different software systems typically used as benchmark applications in the software engineering community, RUBiS[3] and Hadoop[4]. RUBiS is a web application auction site running on the Apache Tomcat[5] web server and serves page requests to hundreds of concurrently simulated clients. Hadoop is a distributed task manager that processes gigabytes of data across multiple nodes. We injected faults into both systems, the virtual machines they are running on, as well as the hosts managing the virtual machines. The faults consumed either processor, memory, disk, database, or network resources in the respective components. We applied generic mitigations that restarted various components, migrated the components to new hosts or virtual machines, or did nothing. The effectiveness of each fault-mitigation pair was analyzed on each system to generate a mapping of faults to mitigations. Using this knowledge, if a fault is later encountered, the best mitigation can be applied with a high chance of success. If a particular mitigation cannot be applied in that instance, then the next best mitigation may be selected, and so forth.

[3] http://rubis.ow2.org/
[4] http://hadoop.apache.org/
[5] http://tomcat.apache.org/

Table 2 - Sensor and kinematic metrics analyzed for convex hull creation

Metric	Type	Status/Measurement
Orientation (CG)	Sensor	Orientation in 3-DOF given by the IMU
Force Torque (feet)	Sensor	Force-torque measurement in X,Y, and Z directions on each ankle
Force Torque (hands)	Sensor	Force-torque measurement in X,Y, and Z directions on each wrist
Orientation (feet)	Sensor	2-DOF orientation (X,Y) of each of the feet
ZMP Location	Kinematics	Current ZMP location in (X,Y,Z)

A naive observer would expect the two systems to have nearly identical fault-mitigation mappings. However, in our testing on software based systems the generated mappings varied greatly in the effectiveness of the mitigations on a particular fault. RUBiS is a typical web application that receives requests from clients and processes results returned from a database. Little processing is being done by RUBiS or its host machine, so even though a processor intensive fault may be using up many cycles in the process or the host, there is no noticeable effect on the request processing rate. As a result, any mitigation applied would result in more down time than if no mitigation were to be applied. Unlike RUBiS, Hadoop is much more processor intensive. Any host level fault that is slowing down the system will adversely affect the completion time for the task. Another major difference is that RUBiS, like many other request driven systems, can easily be restarted to temporarily address fault symptoms, while restarting Hadoop typically means losing already processed results. We expect similar results when applying this method to the complex electro-mechanical platforms Jaemi Hubo/Hubo2+.

The generated fault-mitigation mappings for the latter two software systems offer insight into their respective natures. On the surface, both software systems operate like typical servers, waiting for requests or tasks, and processing them in a timely fashion. However, the way the requests are handled results in faults manifesting in different ways. Even though a fault may be detected, the obvious mitigation is not always the best. This approach to mitigation analysis, will offer valuable insight into the true effectiveness of mitigations when faults occur in Hubo.

3.5 Hubo Mitigations

As in the software case studies, no single mitigation will correct all faults. The effect of each mitigation on a specific fault should be analyzed based on execution time, effectiveness (time to next fault), and risk of damage to the system.

4. EXPECTED RESULTS

It is expected that the recorded metrics will converge and give us a well defined convex hull for normal operating state. Failure states such as ZMP loss or actuator over current is also expected to create a well defined hull due to it's tight correlation with the kinematics and sensor data. Failures that are not tightly correlated other metrics, such as actuator zero or actuator status, are not expected to form a well defined hull. Similar to the software system it is expected that the mitigation methods will very from platform to platform despite having the same faults present.

5. CONCLUSION

Through the failure at the Philadelphia Convention Center it has been shown that there is a need for fault state detection and mitigation. It has been shown that Aniketos system is capable of creating an n-dimensional convex hull describing proper running states and failure states on software system. A plan has been described to apply Aniketos to a physical platform which included injecting faults into the physical system and creating the convex hull from the recorded sensor and state data. The expected results are described in Section 4. and are based on the results from the software testing.

REFERENCES

[1] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan 2003.

[2] J. O. Kephart and R. Das, "Achieving self-management via utility functions," *Internet Computing, IEEE*, vol. 11, no. 1, pp. 40–48, Jan.-Feb. 2007.

[3] S. White, J. Hanson, I. Whalley, D. Chess, and J. Kephart, "An architectural approach to autonomic computing," in *Autonomic Computing, 2004. Proceedings. International Conference on*, May 2004, pp. 2–9.

[4] J. Appavoo, K. Hui, C. A. N. Soules, R. W. Wisniewski, D. D. Silva, O. Krieger, M. A. Auslander, D. Edelsohn, B. Gamsa, G. R. Ganger, P. E. McKenney, M. Ostrowski, B. S. Rosenburg, M. Stumm, and J. Xenidis, "Enabling autonomic behavior in systems software with hot swapping," *IBM Systems Journal*, vol. 42, no. 1, pp. 60–76, 2003.

[5] F. David and R. Campbell, "Building a self-healing operating system," in *Dependable, Autonomic and Secure Computing, 2007. DASC 2007. Third IEEE International Symposium on*, Sept. 2007, pp. 3–10.

[6] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self adaptation with reusable infrastructure," *IEEE Computer*, vol. 37, no. 10, October 2004.

[7] Al-Nashif, Y. and Kumar, A.A. and Hariri, S. and Luo, Y. and Szidarovsky, F. and Qu, G., "Multi-Level Intrusion Detection System (ML-IDS)," in

Autonomic Computing, 2008. ICAC'08. International Conference on, 2008, pp. 131–140.

[8] McAfee, "McAfee-Antivirus Software and Intrusion Prevension Solutions," http://www.mcafee.com/us/.

[9] Symantec, "Symantec - AntiVirus, Anti-Spyware, Endpoint Security, Backup, Storage Solutions," http://www.mcafee.com/us/.

[10] M. Roesch, "Snort - lightweight intrusion detection for networks," in *LISA '99: Proceedings of the 13th USENIX conference on System administration*. Berkeley, CA, USA: USENIX Association, 1999, pp. 229–238.

[11] G. Vigna and R. A. Kemmerer, "Netstat: a network-based intrusion detection system," *J. Comput. Secur.*, vol. 7, no. 1, pp. 37–71, 1999.

[12] M. Brodie, S. Ma, G. Lohman, L. Mignet, M. Wilding, J. Champlin, and P. Sohn, "Quickly finding known software problems via automated symptom matching," in *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, June 2005, pp. 101–110.

[13] Z. Guo, G. Jiang, H. Chen, and K. Yoshihira, "Tracking probabilistic correlation of monitoring data for fault detection in complex systems," in *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, June 2006, pp. 259–268.

[14] Y. Zhao, Y. Tan, Z. Gong, X. Gu, and M. Wamboldt, "Self-correlating predictive information tracking for large-scale production systems," in *ICAC '09: Proceedings of the 6th international conference on Autonomic computing*. New York, NY, USA: ACM, 2009, pp. 33–42.

[15] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase, "Correlating instrumentation data to system states: a building block for automated diagnosis and control," in *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2004, pp. 16–16.

[16] M. Jiang, M. A. Munawar, T. Reidemeister, and P. A. Ward, "System monitoring with metric-correlation models: problems and solutions," in *ICAC '09: Proceedings of the 6th international conference on Autonomic computing*. New York, NY, USA: ACM, 2009, pp. 13–22.

[17] S. Ghanbari and C. Amza, "Semantic-driven model composition for accurate anomaly diagnosis," in *Autonomic Computing, 2008. ICAC '08. International Conference on*, June 2008, pp. 35–44.

[18] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, and A. Fox, "Ensembles of models for automated diagnosis of system performance problems," *Dependable Systems and Networks, International Conference on*, vol. 0, pp. 644–653, 2005.

[19] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," *Autonomic Computing, International Conference on*, vol. 0, pp. 36–43, 2004.

[20] S. Duan and S. Babu, "Guided problem diagnosis through active learning," in *Autonomic Computing, 2008. ICAC '08. International Conference on*, June 2008, pp. 45–54.

[21] E. Stehle, K. Lynch, M. Shevertalov, C. Rorres, and S. Mancoridis, "On the use of computational geometry to detect software faults at runtime," in *Proceeding of the 7th international conference on Autonomic computing*. ACM, 2010, pp. 109–118.

[22] M. Shevertalov, K. Lynch, E. Stehle, C. Rorres, and S. Mancoridis, "Using search methods for selecting and combining software sensors to improve fault detection in autonomic systems," in *Search Based Software Engineering (SSBSE), 2010 Second International Symposium on*. IEEE, 2010, pp. 120–129.

[23] E. Stehle, K. Lynch, M. Shevertalov, C. Rorres, and S. Mancoridis, "Diagnosis of software failures using computational geometry," in *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, nov. 2011, pp. 496 –499.

[24] G. Demiroz and A. Guvenir, "Classification by voting feature intervals," in *9th European Conference on Machine Learning*. Springer, 1997, pp. 85–92.

[25] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox, "Microreboot — a technique for cheap recovery," in *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2004, pp. 3–3.

[26] F. Qin, J. Tucek, J. Sundaresan, and Y. Zhou, "Rx: Treating Bugs As Allergies—A Safe Method to Survive Software Failures," in *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2005, pp. 235–248.

[27] M. Vukobratovic and B. Borovac, "Zero-moment point - thirty five years of its life," *I. J. Humanoid Robotics*, vol. 1, no. 1, pp. 157–173, 2004.

[28] J.-H. Shin and J.-J. Lee, "Fault detection and robust fault recovery control for robot manipulators with actuator failures," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 2, 1999, pp. 861 –866 vol.2.

[29] K. Fujiwara, S. Kajita, K. Harada, K. Kaneko, M. Morisawa, F. Kanehiro, S. Nakaoka, and H. Hirukawa, "Towards an optimal falling motion for a humanoid robot," in *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, dec. 2006, pp. 524 –529.

[30] ——, "An optimal planning of falling motions of a humanoid robot," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, 29 2007-nov. 2 2007, pp. 456 –462.

[31] K. Ogata, K. Terada, and Y. Kuniyoshi, "Real-time selection and generation of fall damage reduction actions for humanoid robots," in *Humanoid*

Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on, dec. 2008, pp. 233 –238.

[32] ——, "Falling motion control for humanoid robots while walking," in *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, 29 2007-dec. 1 2007, pp. 306 –311.

[33] R. Renner and S. Behnke, "Instability detection and fall avoidance for a humanoid using attitude sensors and reflexes," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, oct. 2006, pp. 2967 –2973.

[34] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*, 3rd ed., C. L. Lawson and R. J. Hanson, Eds., 1974.

[35] R. Ellenberg, R. Sherbert, P. Oh, A. Alspach, R. Gross, and J. Oh, "A common interface for humanoid simulation and hardware," in *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, dec. 2010, pp. 587 –592.