UPBOT: A Testbed for Cyber-Physical Systems

Tanya L. Crenshaw and Steven Beyer
University of Portland
Portland, Oregon
crenshaw@up.edu, beyer11@up.edu

Abstract

Developing software for cyber-physical systems presents a unique challenge. These systems are not simply software; they are composed of software running on a collection of machines that present a risk to human safety if anything goes wrong. Researchers want to create languages and tools that aid in the development of secure and fault-tolerant software, but they cannot simply "try out" their ideas on a fighter jet.

This paper describes our modest UPBOT testbed. Pronounced *yoō-pē-bät*, it has three features that aptly comprise a cyber-physical system: networked control, enforceable physical properties, and off-the-shelf components. We offer that UPBOT can be used to effectively test security threats and defenses against cyber-physical systems; it presents multiple points of attack on a programmable, component-based system whose on-board intelligence may maintain safety-critical properties despite malicious attack. Given its low cost and low barrier to entry it may be especially useful to: i) undergraduates interested in learning about the domain; ii) researchers who lack access to oft-unavailable real systems but want to evaluate their solutions for cyber-physical systems.

1 Introduction

We are surrounded by machines. There's the obvious list: laptop, cellphone, mp3 player, e-book. But there are others. Thermostats maintain our office building temperatures. City busses are equipped with a 280 horsepower engine controlled by four processors. All of these machines — from the laptop to the bus engine — are controlled by some kind of software.

Be it the iPhone app developer or embedded systems engineer, it is our hope that anyone writing software for these machines asks these two fundamental questions:

1. Does the code do what it's suppose to do?
2. What does the code do when it breaks?

These two questions practically span the whole of computer science. Generic answers border on the philosophical; they do little to provide solace to the airplane passenger who has been lifted into the air by ninety-four short tons of metal controlled by armies of processors and programs.

Instead, we offer that developers need tools grounded in their specific domain. We ground our own investigations in cyber-physical systems, distributed networks of real-time systems that monitor and control the physical world. They include critical infrastructures such as automobiles and avionics. Their software must do what it should. It must be both fault-tolerant and secure; it must respond to unexpected inputs or malicious attacks in a way that does not risk human safety.

Software engineering practices lack the foundations necessary to reason about these complex systems, unable to uncover satisfying answers to those two fundamental questions. Researchers are currently working towards many visionary solutions. They work towards domain-specific languages [17] [24] and robust software architectures [11] [16]. They work towards middlewares that mitigate complexity [2] [29]. They work towards better verification tools [21]. But the kinds of researchers who are good at developing new software solutions are often far removed from those who are good at building cars or thermostats.

So the need and the domain are at odds with each other. Developers need tools grounded in their specific domain. Researchers want to develop tools, but they cannot simply "try out" their ideas on a fighter jet.

In response, we have created a testbed that comprises a combination of features that reproduce a cyber-physical system: networked control, enforceable physical properties, and off-the-shelf components. Our testbed promises multiple opportunities for evaluating and conducting interesting research in many areas; this paper focuses on the areas of security and artificial intelligence. We offer that UPBOT can be used to effectively test security threats and defenses against cyber-physical systems. Our testbed presents multiple points of attack on a programmable, component-based system whose on-board intelligence may maintain safety-critical properties despite malicious attack.

The rest of the paper is as follows. Section 2 explores the features required for a cyber-physical systems testbed. Section 3 describes our testbed and Section 4 describes our experiments in the testbed. Section 5 describes how the testbed has been used as a teaching tool and Section 6 highlights our lessons learned. Section 7 addresses related work and then we conclude.

2 Defining Testbed Needs

Cyber-physical systems have four characteristics of interest. First, their domains include avionics, automobiles, health management networks, energy, and defense. For these domains, safety is paramount. Second, architectures for these systems are complicated, pieced together from third-party components or components reused from previously deployed systems. Some of these components are fully verified and some are not. Third, because they relate to the physical world, they are subject to unreliable interactions. Sensor input is sporadic or incorrect, and control commands are not always followed with precision. Finally, as networking capability in these systems becomes more ubiquitous, so do the security threats against them.

Offering researchers a meaningful cyber-physical systems testbed requires reproducing the above characteristics in tangible ways. We offer that a testbed requires:

Networked Control. A classical, discrete time control system relates to the physical world; it requires reliable communication, minimal jitter, and hard deadlines [26]. Networked control as seen in cyber-physical systems further requires execution across multiple nodes while mitigating delay, packet loss, and packet corruption. Moreover, networked control presents multiple points of attack by which one may test against security threats.

Enforceable Physical Properties. Cyber-physical systems execute in open contexts in which they interact with dynamic and unpredictable physical environments [5]. Certain physical properties must remain invariant. While simulations are a powerful means to uncovering a system's problems, a physical testbed eliminates assumptions that may hide additional problems [12].

Off-the-Shelf Components. Cyber-physical systems are composed of heterogeneous systems built from off-the-shelf components all talking to each other across multiple networks. As a result, such components must be programmable; defensive and fault-tolerance measures must be built into any software solutions to provide the necessary level of security or assurance.

3 The UPBOT Testbed

Given the previous discussion of the requirements for a meaningful cyber-physical system, we offer our UPBOT testbed. It features networked control of a small fleet of three iRobot Create robots that we have enhanced with wireless communication.

3.1 Physical deployment

The physical deployment of the UPBOT testbed involves four main components.

The **body** of the testbed is deployed on an iRobot Create, the educational version of the popular iRobot Roomba, vacuum excluded. It has no autonomous intelligence; it drives as commanded and it reports sensor data when asked. Its sensors include: i) a left bump and right bump sensor that are activated with the iRobot Create bumps into an obstacle; ii) four cliff sensors on its undercarriage that are activated when the iRobot reaches a precipice like a stair or balcony; iii) an infrared receiver.

The **nerves** comprise a process executing on a gumstix, a small ARM-based motherboard running the Linux Operating System. The nerves communicate with the body over a serial cable using iRobot's comprehensive Serial Command Interface (SCI) protocol. The nerves translate high-level commands to SCI commands and issue them to the body. The nerves also format and convey sensor data to the brain.

The **brain** is a second process executing on the gumstix. It makes local decisions and issues high-level commands to the nerves. Because the gumstix is equipped with a wireless card, the brain can also receive high-level commands from an external supervisor and issue those commands to the body via the nerves. The brain then communicates sensor data wirelessly to the supervisor.

Finally, the **supervisor** is an artificial intelligence component deployed on a desktop machine not directly connected to the body. It makes more complex decisions and uses wirelessly-received sensor data to model the world as perceived by the body.

3.2 Architecture

Having introduced the physical deployment of the UPBOT testbed, we now present its software architecture. Presented in Figure 1, the architecture forms a networked control loop with the various components located on different computing platforms.

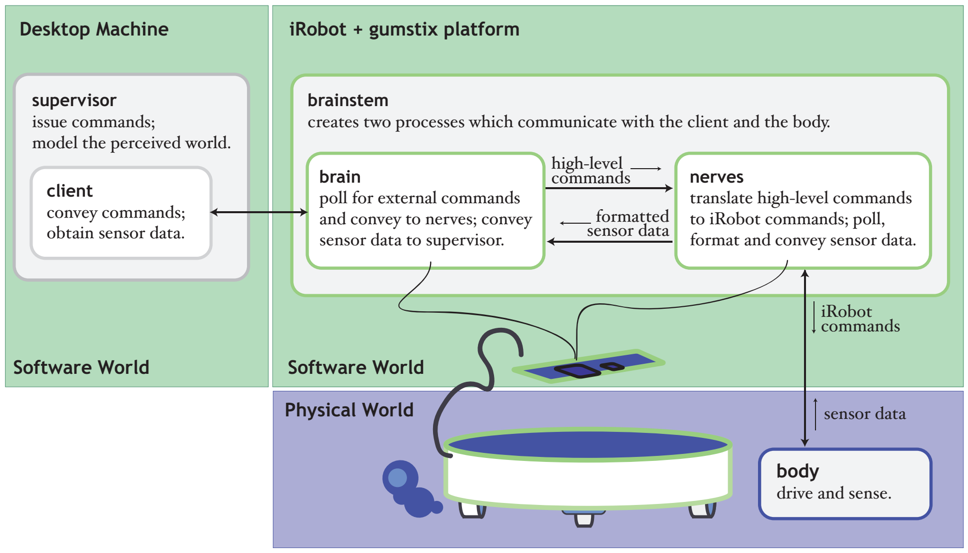
Desktop Machine
supervisor
issue commands; model the perceived world.
client
convey commands; obtain sensor data.
Software World
iRobot + gumstix platform
brainstem
creates two processes which communicate with the client and the body.
brain
poll for external commands and convey to nerves; convey sensor data to supervisor.
high-level commands
formatted sensor data
nerves
translate high-level commands to iRobot commands; poll, format and convey sensor data.
iRobot commands
Software World
Physical World
sensor data
body
drive and sense.

Figure 1: Architecture diagram for the UPBOT testbed. In its networked control loop, sensor data is conveyed from the body all the way up to the supervisor which issues high-level commands that are translated and conveyed to the body.

Starting from the bottom-right of the figure, the nerves continuously poll the body for the most recent sensor data. The nerves translate this sensor data into a friendlier format: ten bits of timestamped data indicate which sensors have been activated. Because both the brain and the nerves are processes executing on the same gumstix platform, the formatted sensor data is conveyed from the nerves to the brain over shared memory.

When the brain obtains the latest sensor data, it may perform two possible operations. It may make a local decision and issue commands back to the nerves. Or it may simply convey the sensor data to the supervisor.

Finally, the brain and the supervisor communicate wirelessly over a socket. The brain relays to the supervisor the formatted sensor data it received from the nerves. The supervisor uses this data to model the perceived world and calculate its next high-level command. This command is issued to the brain, which conveys it to the nerves, which translate it to the appropriate sequence of iRobot commands, completing the loop.

3.3 Meeting the Needs

In Section 2, we offer that for a testbed to be meaningful for cyber-physical systems research, it must posses three important characteristics. We address these three in turn with respect to our UPBOT testbed:

Networked Control. Our UPBOT testbed sees a networked control loop since the supervisor, the brain and nerves, and the body are all located on different computing platforms. Across the three robots in our testbed, this presents multiple points of attack by which one may test against security threats.

Enforceable Physical Properties. Because the UPBOT features a body conducting itself in the physical world, our testbed offers a set of enforceable physical properties. For example, one could write an application that ensures the body never falls off a precipice.

Off-the-Shelf Components. Our UPBOT testbed has no expensive custom hardware. Instead, it features affordable off-the-shelf components: the iRobot Create, an ARM-based gumstix motherboard, a wifistix wireless card, an embedded version of the Linux operating system, and a Debian-based desktop PC. The gumstix motherboard and PC are fully programmable and offer a means to build and evaluate defensive and fault-tolerance measures into the system. The hardware cost for one of our iRobot + gumstix platforms is $456.00.

We do not mean to claim that a bunch of dorky robots are just like a fighter jet. But neither is our testbed just a toy example. We offer that the above characteristics provide a testbed that aptly reproduces a cyber-physical system, but simplifies the overwhelming complexity of a deployed system such that UPBOT is accessible by researchers and undergraduates alike.

Moreover, the iRobot Create is not a toy robot. It is an affordable yet complex mobile robot platform. The Create is developed by iRobot, a company that develops a number of government and industrial robots such as the autonomous Wayfarer battlefield robot currently under research and development for urban reconnaissance missions [28]. Such robots are part of much bigger cyber physical systems in the defense domain.

Finally, while we have offered how the UPBOT testbed reproduces general characteristics of a cyber-physical system, it is also an apt simplification of existing ones. Architectures like the iRobot Wayfarer [28] or the Mars Rover [9] [12] have the same kind of triumvirate as UPBOT: supervisor, brain, and body. A supervisor communicates goals to the brain whose on-board intelligence works to achieve such goals, working with a body layer that controls motors and obtains sensor information. Because we have combined a Create with a gumstix motherboard, we not only have a commercial-grade robot but also computing power enough to make the robot semi-autonomous, able to conduct itself for short periods until it gets new goals from the supervisor. Because UPBOT has the same architecture as other real world systems, we feel that investigations conducted in UPBOT can uncover results that will generalize to other cyber-physical systems.

Certainly UPBOT does not have the scale of larger testbeds such as FIDO [12], a system used by the Jet Propulsion Laboratory to evaluate software solutions for the Mars Rovers. But as successful as such testbeds are at helping experts uncover problems, they are not generally accessible. These bigger testbeds are closed behind doors that are only opened by hefty security clearances or life changing re-employment.

4 Experiments

Our testbed offers multiple opportunities for evaluating and conducting interesting research in many areas; this paper focuses on the areas of security and artificial intelligence. We are currently conducting experiments to evaluate our artificially intelligent supervisor equipped with an episodic memory system [23].

In our current experiments, the supervisor and brain follow a strict alternation. The supervisor issues a command to the brain; the brain responds with the sensor data that resulted from the command. The list of commands that the supervisor may issue to a robot are:

1. Drive forward 630 mm.
2. Turn right 90 degrees.
3. Turn left 90 degrees.
4. Turn right 10 degrees.
5. Turn left 10 degrees.

In the case of the drive forward command, the robot stops if an obstacle is encountered. Since the robot turns in place, there is no need to check for obstacles for the turning commands.

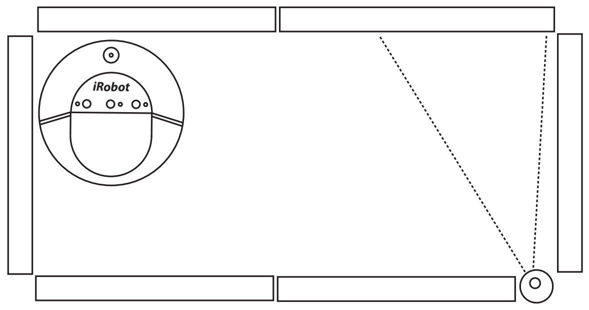

Figure 2: One of our testbed robots navigates a simple maze, 630 mm by 1260 mm in size. The robot learns how to find the infrared transmitter over multiple trials. As the supervisor is developed, the maze will become more complex.

4.1 Learning a Maze in a Safe Environment (ongoing)

Our first ongoing experiment focuses on the learning capability of the supervisor. We want to answer one high-level research question, *How successful is the episodic memory-based supervisor at learning how to navigate the robot through a simple maze?*

Very preliminary results, as shown in Figure 3, demonstrate that we need to improve our learning algorithm, and focus more on accelerated learning. Bottom line: the supervisor is currently a slow learner; it does not do a good job of exploiting past successes. Certainly our results are lamentable in terms of the artificial intelligence community, but they point to the kinds of experiments we can conduct in our testbed. As we continue development of our supervisor, we look forward to improved results. We also look forward to comparing our approach to other AI approaches, such as reinforcement learning.

Though we simulate the robot in a perfect environment and obtain data that displays a smooth learning curve, we cannot say the same for our testbed. It may be obvious, yet it cannot be said enough: Simulating the robot over thousands of perfect, almost instantaneous trials is far different than actually controlling a physical testbed robot that encounters physical obstacles.

4.2 Navigating a Maze in a Threatening Environment (future work)

As part of our future work, we seek to use UPBOT as a testbed for evaluating an episodic memory-based intelligence to detect security threats to a cyber-physical system. Our second experiment will continue investigating the capability of the artificially intelligent supervisor in a threatening environment. We want to answer one high-

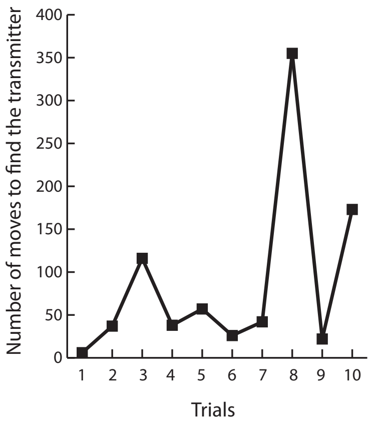

Figure 3: A series of ten trials in which the artificially intelligent supervisor conducts the robot through the simple maze. For each trial, we report the number of moves taken to discover the transmitter. Our results are lamentable, but they point to the kinds of experiments we can conduct in our testbed.

level research question: *"Given that the supervisor has already learned a maze, how successful is the supervisor at navigating the same maze in a threatening environment?"*

An episodic memory architecture enables the supervisor to detect novel situations [23]. If the supervisor has already learned the maze in a safe environment, how will it perform if a malicious agent is also sending wireless commands to the robot, or if a malicious agent has overburdened the brain with useless processes?

5 UPBOT as a Teaching Tool

As much as UPBOT has been built to reproduce important characteristics of a cyber-physical system, it has also been built to be accessible to undergraduates at the University of Portland. To that end, UPBOT has been largely implemented by three undergraduate students, including the second author of this paper.

The development of the testbed has progressed as a series of directed study projects for undergraduates:

- **Fall 2009**. A junior electrical engineering undergraduate interfaces a gumstix motherboard to an iRobot and writes a small program that controls the iRobot Create to drive a pre-planned path and play a song.

- **Spring 2010**. A junior electrical engineering undergraduate develops a client-server application such that one may issue driving commands from a desktop machine wirelessly to the iRobot Create + gumstix mobile robot.

- **Summer 2010.1**. A senior electrical engineering undergraduate increases the robustness of the UPBOT brain software so that it may execute and control the iRobot for multi-hour long experiments.

- **Summer 2010.2**. A sophomore computer science undergraduate develops a device driver for a digital compass so that the UPBOT architecture may leverage not only iRobot Create sensor data, but heading information as well.

- **Summer 2010.3**. A senior computer science undergraduate implements an artificially intelligent supervisor using an episodic memory architecture.

The syllabus for the Spring 2010 directed study course responsible for the wireless connectivity portion of the testbed features three main course objectives:

1. Use cross-compilers for the C programming language.

2. Perform embedded systems development in a UNIX environment using hardware and software tools such as oscilloscopes, digital multi meters, apt-get, tcpdump, objdump, gcc, kermit, grep.

3. Develop and debug wireless networking programs for embedded systems using socket programming.

Course objectives aside, it is unfair to simply hand a motherboard and a robot to an undergraduate and say, *"Go to it"*. The nature of the embedded systems programming demanded by UPBOT is outside the scope of much of any computer science or electrical engineering curriculum. As a result, pair programming [3] has been a key player in developing UPBOT through undergraduate work, with the student always taking the role of the driver.

6 Lessons Learned

We reflect on what we have learned from building the UPBOT testbed, what worked well, and what did not.

6.1 What Worked Well

Leverage What You Know. Given that building a testbed is work-intensive, we did not want to add any unnecessary learning curve to our work. Moreover, we wanted to keep our materials costs low. As the bulk of our experiences were with the Linux Operating System, we sought hardware that could run it.

Document What You Know. Building a testbed involves working many hours to uncover mundane details about serial port and wireless card configurations. It is easy to forget or misplace such details. To that end, we have disciplined ourselves to maintain on-line documentation for interfacing the gumstix to the iRobot Create, including *"Ten steps to building an iRobot + gumstix mobile robot"*, recently requested by a local-area company, Galois, Inc. These resources are publicly available at http://kaju.dreamhosters.com under 'RoboDocs'.

Ask Questions, Then Design. To design a testbed that would be meaningful for a wide variety of cyber-physical systems researchers, we talked to them. Different features stood out as important to different fields:

- **Artificial intelligence**. Important to this field is a hierarchical software architecture that mirrors the kind of hierarchical decision making typically performed. With UPBOT, the brain may make a small set of local decisions while the supervisor may make centralized decisions, or decisions that demand greater computational power.

- **Safety-critical systems**. Important to this field is a networked system architecture built from modules with varying levels of criticality. High-critical modules must continue safely when other modules have failed or communication has been severed. With UPBOT, if the connection between the supervisor and the brain fails, the latter safety-critical module may continue making local decisions to keep the body safe.

6.2 What Did Not Work Well

Take Care in Choosing Hardware. The gumstix hardware is somewhat challenging to use. First, the gumstix connex was discontinued soon after purchasing it. Second, gumstix documentation is offered only in the form of on-line articles, FAQs, a mailing list, and a sparse wiki. Given a time-machine, we would choose a different motherboard to interface to the iRobot Create.

Use Elegant Interprocess Communication. Given the architecture for the UPBOT testbed presented in Figure 1, many activities are happening concurrently. For example, the brain and the nerves are continuously passing sensor data and commands back and forth. It is nowhere near elegant; it requires a lot of tedious implementation details related to interprocess communication with sockets and pipes. Bugs are inevitable.

7 Related Work

We have built the UPBOT testbed to offer researchers a meaningful testbed for evaluating their software solutions in the cyber-physical systems domain. Other work seeks to achieve this goal. The Convergence Lab at the University of Illinois at Urbana-Champaign [15] focuses on the challenges of networked control and features a small fleet of remote-controlled cars. Unlike our testbed, the cars have no on-board intelligence and are dependent on a centralized trajectory planner which directs the cars. Larger-scale testbeds are also available [14]. We appreciate the scale of such testbeds, but we offer that an advantage to UPBOT's scale is a lower barrier to entry. Like UPBOT, other testbeds focus on education, including [19] which sees iRobot Creates controlled by the now-retired gumstix connex motherboard. The popular robotics platform Player/Stage is installed on the gumstix to empower its programmers with a high-level API that makes it easy to control the robot. However, because we wanted UPBOT to be useful for both computer science and electrical engineering undergraduates, we control the robot with lower-level C programming.

It is not always feasible or desirable to build a physical testbed. Simulation-based testbeds also offer value. One approach is to specify a hardware-in-the-loop (HIL) test environment; the hardware to be controlled is simulated in real-time and the control software is tested against the simulation. Much of an automobile is controlled by software, so industry relies on HIL to evaluate systems such as braking, steering, or emissions control [7]. Automobiles are not the only domain. Armbruster et al. have developed an HIL for bulk power systems control [1]. Though a useful approach to evaluating a system, HILs are limited by the combinatorial complexity of the hardware being simulated.

Especially in the area of security, it is sometimes necessary to build a testbed that offers realistic scenarios. Both [4] and [25] offer a traditional large-scale IT system in which researchers can deploy and evaluate network services. As of 2006, PlanetLab [25] offered a testbed comprising 694 nodes in 35 countries. The scale of such projects is impressive; UPBOT does not boast such a scale. Still, compared to traditional IT systems, many cyber-physical systems have different needs for security their defenses [8]. Cyber-physical systems have comparatively simpler network dynamics but they must protect the system under control; an ideal example is the power substation that continues to deliver power to its customers despite a malicious attack.

In traditional IT systems, many defenses against malware have been based on detecting *known* malware. In the days of DOS-based viruses, malware detection schemes boiled down to string matching algorithms that searched for the current set of known viruses [22]. Since then, malware has become more complex: encrypted and polymorphic viruses [6], worms, bots, and rootkits [27]. Today's malware detectors look for malicious code fragments and signatures instead of whole programs. Yet for cyber-physical systems, security rarely means looking for a signature of a known trojan horse

Instead, what perhaps may make more sense for defending cyber phyiscal systems is the behavior blocking approach. Behavior blocking software may flag suspicious code or monitor real-time program behaviour for sequences of malicious actions. It then isolates the offender in a sandbox; an administrator determines if it should be removed or allowed to run [20].

To block malicious behavior, it must first be detected. This is sometimes described in the literature as *anomaly detection* and has been developed in areas ranging from machine learning to statistics [10]. Often, these systems build models of normal data and detect deviations. As described in Section 4.2 We seek to use UPBOT as a testbed for evaluating an episodic memory-based intelligence to detect threats to a cyber-physical system. Other works have seen anomaly detection in similar domains, including [13] which applies anomaly detection to telemetry data for early detection of component failures in spacecraft and [18] which applies anomaly detection to network intrusion.

8 Conclusion

Developing software for cyber-physical systems presents a unique challenge. These systems are composed of software running on a collection of machines that present a risk to human safety if anything goes wrong. It is difficult to evaluate software solutions for cyber-physical systems because so much of these systems exist in the physical world.

UPBOT is not a fighter jet, nor is it just a toy example that focuses on clean carpets. We offer that UPBOT can be used to effectively test security threats and defenses against cyber-physical systems; it presents multiple points of attack on a programmable, component-based system whose on-board intelligence may maintain safety-critical properties despite malicious attack.

For more information on the technical implementation details for UPBOT, please refer to http://kaju.dreamhosters.com under 'RoboDocs'. We respectfully request that those who use our documentation to build an iRobot Create + gumstix mobile robot platform please cite this paper.

9 Acknowledgments

This work was made possible by the University of Portland Arthur Butine Faculty Development Fund. We are grateful to those who provided comments that made this paper better: Joe Hendrix, Joe Hoffbeck, Steve Miller, Jelena Mirkovic, Andrew Nuxoll and the anonymous reviewers of the CSET'10 workshop.

References

[1] Austin Armbruster, Matt Ryan, Xiaoqing Frank Liu, Ying Cheng, and Bruce M. McMillin. Hardware/software co-design for power system test development. In *WISER*, pages 83–88, 2004.

[2] Girish Bagliga, Scott Graham, Lui Sha, and P. R. Kumar. Etherware: Domainware for wireless control networks. *Object-Oriented Real-Time Distributed Computing, IEEE International Symposium on*, 0:155–162, 2004.

[3] K. Beck. *eXtreme Programming Explained: Embrace Change*. Addison Wesley Longman, 2000.

[4] Terry Benzel, Robert Braden, Dongho Kim, and Cliford Neuman. Experiences with deter: A testbed for security research. In *In 2nd IEEE Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities (TridentCom)*, 2006.

[5] Azer Bestavros. Towards safe and scalable cyber-physical systems. In *NSF Workshop on Cyber-Physical Systems*, 2006.

[6] Matt Bishop. *Computer Security: Art and Science*. Addison Wesley Professional, 2003.

[7] Manfred Broy. Challenges in automotive software engineering. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 33–42, New York, NY, USA, 2006. ACM.

[8] Alvaro A. Cardenas, Saurabh Amin, and Shankar Sastry. Secure control: Towards survivable cyber-physical systems. In *ICDCSW '08: Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems Workshops*, pages 495–500, Washington, DC, USA, 2008. IEEE Computer Society.

[9] Rebecca Castano, Tara Estlin, Daniel Gaines, Andres Castano, Caroline Chouinard, Ben Bornstein, Robert C. Anderson, Steve Chien, Alex Fukunaga, and Michele Judd. Opportunistic rover science:

Finding and reacting to rocks, clouds and dust devils. In *IEEE Aerospace Conference*, March 2006.

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):1–58, 2009.

[11] Bruce Douglass. *Real-Time Design Patterns*. Addison-Wesley Publishing Company, 2003.

[12] Tara A. Estlin, Daniel M. Gaines, Caroline Chouinard, Rebecca Castao, Benjamin Bornstein, Michele Judd, Issa A. D. Nesnas, and Robert C. Anderson. Increased mars rover autonomy using AI planning, scheduling and execution. In *2007 IEEE International Conference on Robotics and Automation, ICRA 2007, 10-14 April 2007, Roma, Italy*, pages 4911–4918. IEEE, 2007.

[13] Ryohei Fujimaki, Takehisa Yairi, and Kazuo Machida. An approach to spacecraft anomaly detection problem using kernel feature space. In *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 401–410, New York, NY, USA, 2005. ACM.

[14] Annarita Giani, Gabor Karsai, Tanya Roosta, Aakash Shah, Bruno Sinopoli, and Jon Wiley. A testbed for secure and robust scada systems. *SIGBED Rev.*, 5(2):1–4, 2008.

[15] Scott Graham and P. R. Kumar. The convergence of control, communication, and computation. *Proceedings of PWC 2003: Personal Wireless Communications. Lecture Notes in Computer Science*, 2775, 2003.

[16] Cheolgi Kim, Mu Sun, Sibin Mohan, Heechul Yun, Abdullah Al-Nayeem, Lui Sha, and Tarek Abdelzaher. A framework for the safe interoperability of medical devices in the presence of connection failures. In *International Conference on Cyber-physical Systems (ICCPS)*, Stockholm, Sweden, April 2010.

[17] David Kitchin, Adrian Quark, William Cook, and Jayadev Misra. The Orc programming language. In *Proceedings of FMOODS/FORTE 2009*, June 2009.

[18] Ar Lazarevic, Aysel Ozgur, Levent Ertoz, Jaideep Srivastava, and Vipin Kumar. A comparative study of anomaly detection schemes in network intrusion detection. In *In Proceedings of the Third SIAM International Conference on Data Mining*, 2003.

[19] Maja J Mataríc, Nathan Koenig, and David Feil-Seifer. Materials for enabling hands-on robotics and stem education. In *AAAI Symposium on Robots and Robot Venues: Resources for AI Education*, March 2007.

[20] E Messner. Behavior blocking software repels new viruses. *Network World*, January 2002.

[21] Steven P. Miller, Elise A. Anderson, Lucas G. Wagner, Michael W. Whalen, and Mats P.E. Heimdahl. Formal verification of flight critical software. In *Proceedings of the AIAA Guidance, Navigation and Control Conference and Exhibit*, August 2005.

[22] Carey Nachenberg. Computer virus coevolution. *Communications of the ACM*, 40(1), January 1997.

[23] Andrew Nuxoll and John E. Laird. Extending cognitive architecture with episodic memory. In *AAAI*, pages 1560–1564, 2007.

[24] John Peterson, Paul Hudak, and Conal Elliott. Lambda in motion: Controlling robots with haskell. In *PADL*, pages 91–105, 1999.

[25] Larry Peterson, Andy Bavier, Marc E. Fiuczynski, and Steve Muir. Experiences building planetlab. In *In Proceedings of the 7th USENIX Symp. on Operating Systems Design and Implementation (OSDI*, 2006.

[26] Charles L. Phillips and H. Troy Nagle. *Digital Control System Analysis and Design*. Prentice Hall, 1994.

[27] Willam Stallings and Lawrie Brown. *Computer Security: Principles and Practice*. Pearson Prentice Hall, 2008.

[28] Brian Yamauchi. The wayfarer modular navigation payload for intelligent robot infrastructure. In *SPIE, Vol. 5804, 85*, March 2005.

[29] Yuanfang Zhang, Christopher Gill, and Chenyang Lu. Reconfigurable real-time middleware for distributed cyber-physical systems with aperiodic events. In *ICDCS '08: Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems*, pages 581–588, Washington, DC, USA, 2008. IEEE Computer Society.