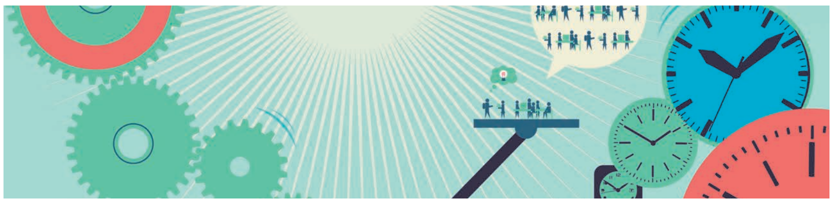
Software Adaptation for an Unmanned Undersea Vehicle

Avi Pfeffer, Curt Wu, Gerald Fry, Kenny Lu, Steve Marotta, and Mike Reposa, Charles River Analytics

Yuan Shi, T.K. Satish Kumar, and Craig A. Knoblock, University of Southern California Information Sciences Institute

David Parker, Irfan Muhammad, and Chris Novakovic, University of Birmingham

// Most current software systems are not adaptable, making them less capable of achieving their objectives. We are developing a method to optimize software for new environments automatically. An independent evaluation has demonstrated that this method adapts to degraded sensors, changing environmental conditions, and a loss of power. //



UNMANNED UNDERSEA VEHICLES (UUVs) are designed to carry out challenging missions in changing environments. To maximize their effectiveness, these vehicles should adapt to system failures (such as battery loss) and environmental changes (such as a force on the UUV). Since it is expensive to develop UUVs, it is also desirable to increase their lifespan by enabling their software to be able adapt to ecosystem changes such as upgraded sensors.

Digital Object Identifier 10.1109/MS.2018.2886815
Date of publication: 22 February 2019

In our Probabilistic Representation of Intent Commitments to Ensure Software Survival (PRINCESS) project, which was part of the Defense Advanced Research Projects Agency (DARPA) Building Resource Adaptive Software Systems (BRASS) program, we are developing methods to adapt the UUV's software for all these purposes. Our sensor adaptation accommodates new and upgraded sensors as well as compensates for sensor degradation while the UUV is on a mission. Our control adaptation responds to online system failures and environmental changes in real time; we use probabilistic verification techniques to ensure that these adaptations do not result in software behavior that is dangerous for the UUV.

Our recent work on PRINCESS has involved two scenarios for a REMUS 600 UUV. The first scenario involves degrading a Doppler velocity log sensor used for navigation and a simultaneous perturbation to the environment in the form of a high current. Our adaptation reconstructs an estimate of the sensor signal from other sensors and adjusts the parameters of the navigation system's Kalman filter to account for the increased noise and environmental perturbation. In the second scenario, the UUV undergoes a catastrophic loss of battery power while on a reconnaissance mission for an object on the ocean floor. Our adaptation reconfigures the UUV's path planner to generate a path that searches as much of the region as possible while still bringing the UUV home safely without running out of power.

Adaptation Methods

Sensor Adaptation

The ability to detect and adapt to sensor failures has a number of benefits. In the UUV domain, our approach significantly reduces the time and effort required for software maintenance when a sensor fails or is replaced. Instead of changing the software itself, it invokes a joint detection and adaptation (JDA) module.

Our sensor adaptation assumes that sensor values among a subset of sensors are correlated, which is often true in real-world systems.[1] While JDA uses several techniques from machine learning (ML), its real power stems from its novel constraint-based framework in which these ML techniques are embedded. We note that a naïve application of ML techniques to reconstruct one sensor value from other ones is not viable because multiple sensors can fail at the same time. Instead, we first learn a substrate set of constraints in JDA. These constraints are in the general form $(y_n - f_n(z_n))^2 \leq \varepsilon_n{}^2$, where y_n is the target sensor value at time t, z_n is a set of input sensor values at time less than or equal to t, and $f_n()$ is a reconstruction function.

Ideally, the reconstruction functions allow for the accurate reconstruction of failed sensor values, are comprehensive enough to be able to adapt to many kinds of failures, and are generally easy to understand. In JDA, we learn the functions by using a blend of ML methods and other heuristic methods that first identify the variables of interest. For example, casting this as a least absolute shrinkage and selection operator (LASSO) problem[2] can help us to first identify a sparse set of variables that determine the value of a target sensor up to a certain level of accuracy. After these variables are determined, we can use ML techniques to learn the actual reconstruction function.[3] Similar LASSO problem instances can be used subsequently to identify a second, third, or generally the kth set of relevant variables that minimizes overlaps with the previous sets of variables.

Such a substrate of constraints is viable for detecting and adapting to multiple sensor failures. First, a violated constraint indicates a sensor failure—in particular, that at least one of the sensors involved in that constraint has failed. Solving an integer linear program identifies a minimum set of such failed sensors that account for all violated constraints.[4] After sensors are determined to have failed or to be in working condition, reconstruction of failed sensor values, i.e., adaptation, begins. To reconstruct the value of a failed sensor, we simply find a constraint with minimum ε_n in which y_n is the target sensor value and all sensors in z_n are deemed to be in working condition. As a natural consequence of our constraint-based method, ε_n can also be used to estimate how good our adaptation is.

Control Adaptation and Verification

The goal of control adaptation is to adjust the UUV software in real time in response to perturbations (such as a loss of battery power) and environmental changes (such as a change of current). In PRINCESS, we work with legacy software components that do not have any controllable parameters with understood semantics, such as the UUV's Kalman filter and path-planner components. Therefore, we must make those components adaptive by increasing their range of behavior and synthesizing control parameters. We must also learn the meaning of those control parameters; in other words, we must understand how different settings of the controls enable the component to achieve its intent in different situations.

Our method uses a combination of program transformation and ML. First, we introduce variable behavior into the software component. Beginning with a component with a given set of inputs and fixed behaviors, we analyze the code to identify candidates for variation, such as constants or inequalities. We then parameterize these candidates, for example, by replacing a constant with a control variable or adding a control variable to one side of a loop inequality. Next, we transform the interface of the component to take the control parameters as input to produce a transformed component that is ready for adaptation.

The next step is to learn how to set the values of the controls in each situation. Via a simulator, we generate a large data set of inputs, environment variables, and controls and then run the software component and evaluate the intent of the component. PRINCESS uses this data set to train a feed-forward neural network, which then identifies the optimal value of the controls for each setting of the inputs and environment variables. This creates a supervised learning problem in which we learn mapping from the state of the inputs and environment variables to the optimal controls.

The final step is to combine the learned optimization policy with the transformed component to produce an optimizing component. Given the values of the inputs and environment variables, the optimization policy produces values for the controls that are fed into the transformed component. This optimizing component functions in a transformed software system alongside a monitor that keeps track of the state of environment variables and passes them to the optimizing component. Further details on the program's transformation and optimization can be found in Fry et al.[5]

Optimizing component controls using ML techniques could produce dangerous adaptations and may be difficult to trust. To assure the safety and reliability of our control adaptations, we employ formal verification techniques. In particular, we use probabilistic model checking, a technique for producing guarantees about the quantitative aspects of a system's runtime behavior, such as execution time, energy usage, or the probability of failure. This approach is based on the systematic construction and numerical analysis of a stochastic model, which yields a probabilistic guarantee on a system property that was formally specified in temporal logic.

In this article, we deploy verification at the runtime, automatically building and solving models that represent the execution of the current mission plan. Currently, this process focuses on the path-planning component of the UUV. Each time an adaptation occurs, it generates a new path plan for the UUV's mission, and we verify whether the adaptation can be applied safely. If an adaptation is considered to be unsafe, PRINCESS tightens the constraints on the adaptation requirements (i.e., reduces the allowed power usage) and generates a new adaptation candidate. This process repeats for a fixed number of times until the optimizer finds an adaptation with a probabilistic guarantee of success or until the feedback loop reaches the repetition threshold. In the latter case, PRINCESS will either proceed with the most recent adaptation candidate or return home, depending on its policy.

In this context, the primary risk of a mission failing is the possibility that the vehicle is stranded and unable to return home if the battery runs out of fuel. Therefore, a key aspect of the model used for verification is the UUV's energy consumption. Due to environmental uncertainty, this aspect of its behavior needs to be modeled stochastically and is the main reason that we produce a probabilistic guarantee of mission success (the likelihood of completing the current search process and then safely returning home). The model we construct is a discrete-time Markov chain whose state incorporates both the current position of the UUV in its mission and its battery level.

We use an adapted version of the PRISM probabilistic verification software[6]; in particular, we connected to its Java application programming interface, which allows us to construct models on the fly using a generative model interface. We built upon earlier PRISM-based methods for producing verified navigation plans for mobile robots.[7] The portion of the model that captures how energy usage varies with the UUV's location and speed is offline, using traces generated from simulated behavior of the UUV. The result is a parameterized model that can be reconstructed at the runtime, depending on the current status of the UUV at a given point in the mission.

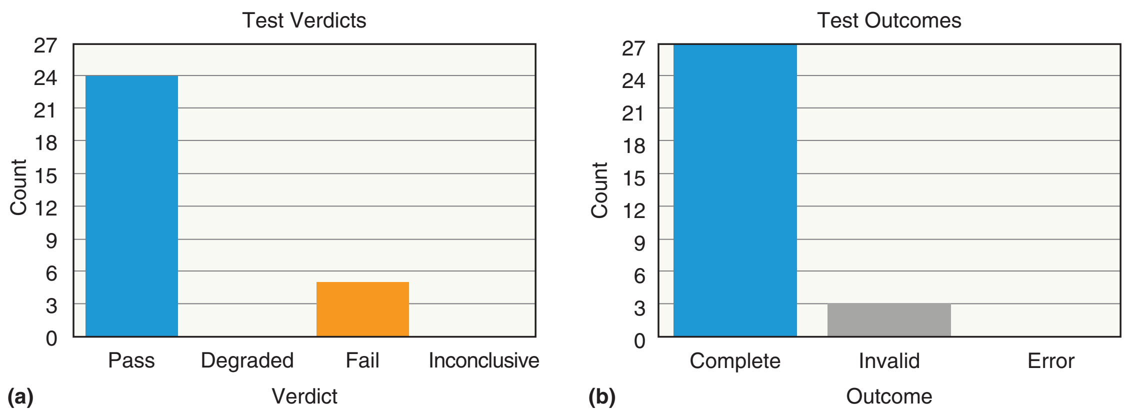


FIGURE 1. The results of the MIT Lincoln Laboratory evaluation of the sensor adaptation approach. The (a) test verdicts and (b) test outcomes.

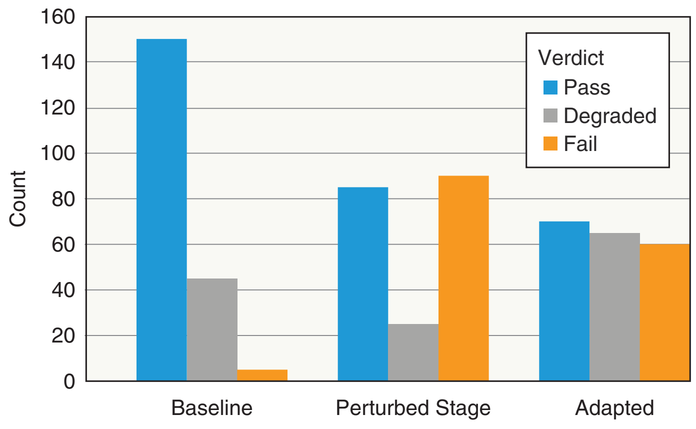


FIGURE 2. The results of the MIT Lincoln Laboratory evaluation of the control adaptation and verification approach.

Results

Our adaptation approaches were independently evaluated by the Massachusetts Institute of Technology (MIT) Lincoln Laboratory as part of the DARPA BRASS Program. The sensor adaptation approach was evaluated under a scenario in which the UUV must navigate from a starting position to a specified destination. Results are shown in Figure 1. During transit, the UUV encountered a water current and experienced a sensor failure. For each scenario, the evaluators recorded a "pass" verdict if the adapted UUV ended within 75 m from the destination. Only scenarios in which the nonadapted system failed to reach this threshold were considered.

The control adaptation and verification approach were evaluated under scenarios in which the UUV had to search a rectangular region of the sea floor to find an object. We simulated battery failures in each scenario. The UUV had to find the object and return to its starting point, and it had to adapt its search path when energy perturbations occurred. Figure 2 shows the results of the same scenarios run in the baseline (no failures), perturbed (no adaptation with battery failures), and adapted (with battery failures and adaptation) stages. The verdicts are defined as

- pass: object found and UUV returns
- degraded: object not found and UUV returns
- fail: the UUV depletes its energy before it can return.

Conclusions

A UUV provides an ideal platform to study many aspects of software adaptation. In PRINCESS, we have successfully demonstrated an adaptation to upgraded and degraded sensors, system failures, environment changes, and new architecture. In our ongoing work, our goal is to generalize our methods beyond UUVs to other software systems. Our control adaptation, for example, uses general techniques of program transformation and ML that could, in principle, be applied to a wide variety of systems in different programing languages. We also aim to smooth out and automate as much of the process as possible, by which a legacy code base transforms into an adaptive code base. These developments have the potential to not only increase the life of software but also make the software behave more appropriately in its new environment than the original software.

During our experimentation with intents for the navigation system, we discovered that the intent of the Kalman filter does not map directly to the operational intent of the UUV's navigation system. In contrast, the path planner's intent to maximize area coverage while restricting energy consumption is analogous with maximizing its probability of finding a randomly placed object within the area, thus yielding much better results even though the optimization approach was the same. This result underscores the notion that proper intent specification ultimately drives the optimization of the system regardless of the approach used for implementing optimization. In the future, we will work with subject matter experts to improve our precision in defining operationally relevant intents.

Additionally, program transformations can introduce a large number of control parameters to the program. In the case of the Kalman filter, the transformation increases the number of inputs by at least an order of magnitude. This implies that we need to search an exponentially large space of possible input combinations. While our ML models enable us to represent this space relatively compactly, we still need to generate a large amount of data to train the model. For the relatively simple software components we worked on, we were able to train a basic model effectively. As the components become more complex, we will need a more detailed understanding of the parameter space and more intelligent model designs.

Finally, these experiments and results further highlight the complementary roles that optimization and verification play in our adaptation process. Without the verifier, an overeager optimizer may choose parameters that would further damage an already perturbed UUV. Conversely, a verifier without an optimizer, while robust and fault tolerant, would be brittle to new scenarios in which prior knowledge is lacking or nonexistent. Overall, both are necessary to provide meaningful and practical adaptations.

Acknowledgment

This article is based on work supported by the U.S. Air Force and DARPA under contract FA8750-16-C-0045. The views, opinions, or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the U.S. Department of Defense or the U.S. Government.

References

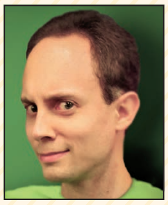
1. E. Elnahrawy and B. Nath, "Context-aware sensors," in *Proc. European Workshop Wireless Sensor Networks*, 2004, pp. 77–93.

ABOUT THE AUTHORS

AVI PFEFFER is a chief scientist at Charles River Analytics. His research interests include probabilistic reasoning, machine learning, and computational game theory. Pfeffer received a Ph.D. in computer science from Stanford University. Contact him at apfeffer@cra.com.

KENNY LU is a scientist at Charles River Analytics. His research interests include program analysis, Bayesian inference, and reinforcement learning. Lu received a B.S. in mathematical science from Carnegie Mellon University. Contact him at klu@cra.com.

CURT WU is a chief software engineer at Charles River Analytics. His research interests include cyber security, applied machine learning, and software resilience. Wu received an M.S. in computer science from Boston University. He is a Member of the IEEE. Contact him at cwu@cra.com.



STEVE MAROTTA is a senior software engineer at Charles River Analytics. His research interests include machine learning, code generation, and programming languages. Marotta received a B.A. in computer science from Drew University. He is a Member of the IEEE. Contact him at smarotta@cra.com.

GERALD FRY is a scientist at Charles River Analytics. His research interests include distributed systems, real-time resource management, and virtualization. Fry received an M.S. in computer science from Boston University. Contact him at gfry@cra.com.

MIKE REPOSA is a senior software engineer at Charles River Analytics. His research interests include adaptive systems, distributed systems, and systems integration. Reposa received a B.S. in computer engineering technology from Northeastern University. Contact him at mreposa@cra.com.

ABOUT THE AUTHORS

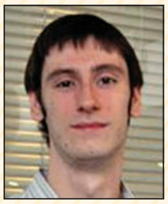
YUAN SHI is a Ph.D. student at the University of Southern California. His research interests include machine learning, optimization, and computer vision. Shi received a B.Eng. in computer science from Sun Yat-sen University. Contact him at yuanshi@usc.edu.

DAVID PARKER is a reader at the University of Birmingham. His research interests include probabilistic verification and he is the lead developer of the PRISM model checking tool. Parker received a Ph.D. in computer science from the University of Birmingham. Contact him at d.a.parker@cs.bham.ac.uk.

T.K. SATISH KUMAR is a research assistant professor at the University of Southern California. His research interests include constraint reasoning, probabilistic reasoning, planning, and scheduling. Kumar received a Ph.D. in computer science from Stanford University. Contact him at tkskwork@gmail.com.

IRFAN MUHAMMAD is a Ph.D. student at the University of Birmingham. His research interests include probabilistic verification. Muhammad received a B.S. in computer science from the University of Birmingham. Contact him at irfan.mu3@gmail.com.

CRAIG A. KNOBLOCK is a research professor at the University of Southern California. His research interests include information integration and the semantic web. Knoblock received a Ph.D. in computer science from Carnegie Mellon University. He is a Senior Member of the IEEE. Contact him at knoblock@isi.edu.



CHRIS NOVAKOVIC is a research fellow at the University of Birmingham. His research interests include computer security and distributed systems. Novakovic received a Ph.D. in computer science from the University of Birmingham. Contact him at c.novakovic@cs.bham.ac.uk.

2. R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. Roy. Statist. Soc. Ser. B Methodol*, vol. 58, no. 1, pp. 267–288, 1996.
3. N. M. Nasrabadi, "Pattern recognition and machine learning," *J. Electron. Imag.*, vol. 16, no. 4, 2007, Art. no. 049901.
4. H. Marchand, A. Martin, R. Weismantel, and L. Wolsey, "Cutting planes in integer and mixed integer programming," *Discrete Appl. Math.*, vol. 123, no. 1–3, pp. 397–446, 2002.
5. G. Fry et al., "Adapting autonomous ocean vehicle software systems to changing environments," presented at the Oceans Conf. and Expo., 2018.
6. M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Computer Aided Verification* (LNCS, vol. 6806), G. Gopalakrishnan and S. Qadeer, Eds. Berlin: Springer, 2011, pp. 585–591.
7. B. Lacerda, D. Parker, and N. Hawes, "Multi-objective policy generation for mobile robots under probabilistic time-bounded guarantees," in *Proc. 27th Int. Conf. Automated Planning Scheduling (ICAPS)*, 2017, pp. 504–512.