Software Adaptation for an Unmanned Undersea Vehicle

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

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

David Parker, Irfan Muhammad, Chris Novakovic
University of Birmingham

Most current software systems cannot adapt to changing hardware requirements or changing operational environments. This lack of adaptivity shortens the life of the software and makes it less capable of achieving its mission. We are developing Probabilistic Representation of Intent Commitments to Ensure Software Survival (PRINCESS) to enable software to adapt to both hardware changes and changing environments. We are developing methods to automatically optimize software for new environments, turning non-adaptive code into optimizable adaptive code. We are also developing sensor adapters that enable us to maintain software function despite changing or failing sensors. We are implementing PRINCESS on the navigation system of an unmanned undersea vehicle. An independent evaluation has demonstrated PRINCESS's ability to adapt to degraded sensors, changing environmental conditions, and loss of power.

Introduction

Unmanned undersea vehicles (UUVs) are designed for challenging missions in changing environments. To maximize their effectiveness, these vehicles should adapt to system failures (such as loss of a battery) and changes to the environment (such as a force on the UUV). Since the development of UUVs is expensive, it is also desirable to increase their lifespan by making their software adapt to ecosystem changes like upgraded sensors.

In our Probabilistic Representation of Intent Commitments to Ensure Software Survival (PRINCESS) project, part of DARPA's 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.

In our recent work on PRINCESS, we have worked on two scenarios involving a REMUS 600 UUV. The first scenario involves degradation of a Doppler velocity log (DVL) 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 environment 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. Particularly, in the UUV domain, it significantly reduces the time and effort required for software maintenance when a sensor fails or is replaced. Instead of changing the software itself, our approach 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 a novel constraint-based framework in which these ML techniques are embedded. We note that a naive application of ML techniques to reconstruct one sensor value from other sensor values is unviable since multiple sensors can fail at the same time. Instead, in JDA, we first learn a substrate set of constraints. These constraints have 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 $\leq t$, and $f_n()$ is a reconstruction function.

The reconstruction functions ideally allow for the accurate reconstruction of failed sensor values, are comprehensive enough to be able to adapt too many kinds of failures, and are easy to understand in general. In JDA, we learn them using a blend of ML methods and other heuristic methods that first identify the variables of interest. For example, casting a LASSO problem [2] instance can help us 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, ML techniques can be used to learn the actual reconstruction function [3]. Similar LASSO problem instances can subsequently be used to identify a second, third, or generally, the k^{th} set of relevant variables that minimizes overlap with the previous sets of variables.

Such a substrate of constraints can be viably used for detecting and adapting to multiple sensor failures. First, a violated constraint indicates a sensor failure and, in particular, indicates that at least one of the sensors involved in that constraint has failed. A minimum set of such failed sensors that account for all violated constraints is identified by solving an Integer Linear Program [4]. After sensors are deemed to have failed or to be in a working condition, reconstruction of failed sensor values, i.e., adaptation is invoked. 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 a working condition. As a natural consequence of our constraint-based method, ε_n can also be used as an estimate of how good our adaptation is.

Control Adaptation and Verification

The goal of control adaptation is to adapt the UUVs software in real time in response to perturbations (like loss of battery power) and environmental changes (like 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 machine learning. First, we introduce variable behavior into the software component. Beginning with a component with a given set of inputs and fixed behaviors, we first 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. We then transform the interface of the component to take the control parameters as input to produce a transformed component ready for adaptation.

The next step is to learn how to set the values of the controls in each situation. To achieve this, we generate, via a simulator, a large dataset of inputs, environment variables, and controls, and run the software component and evaluate the intent of the component. PRINCESS uses this dataset 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, where we learn a 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 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 transformation and optimization can be found in [5].

Optimizing component controls using machine learning techniques could produce dangerous adaptations and may be difficult to trust. To provide assurances about the safety and reliability of our control adaptations, we employ *formal verification* techniques. In particular, we use *probabilistic model checking*, which is a technique for producing guarantees about 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 formally specified in temporal logic.

In this work, we deploy verification at runtime, automatically building and solving models representing the execution of the current mission plan. Currently, this focuses on the path planning component of the UUV. Each time that an adaptation occurs, generating a new path plan for the UUV's mission, we apply verification to check if the adaptation can be applied safely. If the verifier deems an adaptation to be unsafe, PRINCESS tightens the constraints on the adaptation requirements (i.e., reduces 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 policy.

The primary risk of a mission failing in this context is the possibility of the vehicle becoming stranded due to the battery depleting and it being unable to return home. So, a key aspect of the model used for verification is the energy consumption of the UUV. Due to environmental uncertainty, this aspect of its behavior needs to be modelled stochastically, and this 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 the battery level.

We use an adapted version of the PRISM probabilistic verification software [6], in particular connecting to its Java API, which allows models to be constructed on the fly using a generative model interface. We build 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 learnt offline, using traces generated from simulated behavior of the UUV. The result is a parameterized model that can be reconstructed at runtime, depending on the current status of the UUV at a given point in the mission.

Results

Our adaptation approaches were independently evaluated by MIT Lincoln Laboratory as part of the DARPA BRASS Program. The sensor adaptation approach was evaluated under scenarios in which the UUV must navigate from a starting position to a specified destination. Results are shown in Figure 1. During transit, the UUV encounters a region of water current and experiences a sensor failure. For each scenario, the evaluators recorded a *Pass* verdict if the adapted UUV ended within 75m from the destination. Only scenarios in which the non-adapted system failed to reach this threshold were considered.

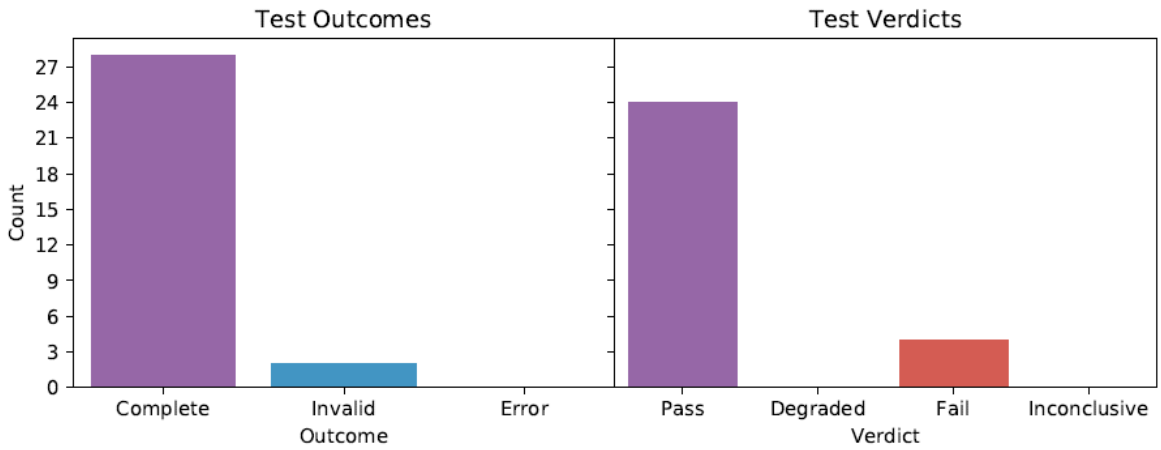

Figure 1: Results of MIT Lincoln Laboratory evaluation of sensor adaptation approach

The control adaptation and verification approach was evaluated under scenarios in which the UUV must search a rectangular region of the sea floor to find an object. We simulate battery failures in each scenario. The UUV must find the object and return to its starting point, and it must adapt its search path when energy perturbations occur. Figure 2 shows the results of the same scenarios run in Baseline (no failures), Perturbed (no adaptation with battery failures), and Adapted (with battery failures and adaptation) stages. The verdicts are defined as follows:

- Pass – object found and UUV returns
- Degraded – object not found and UUV returns
- Fail – the UUV depletes its energy before it can return

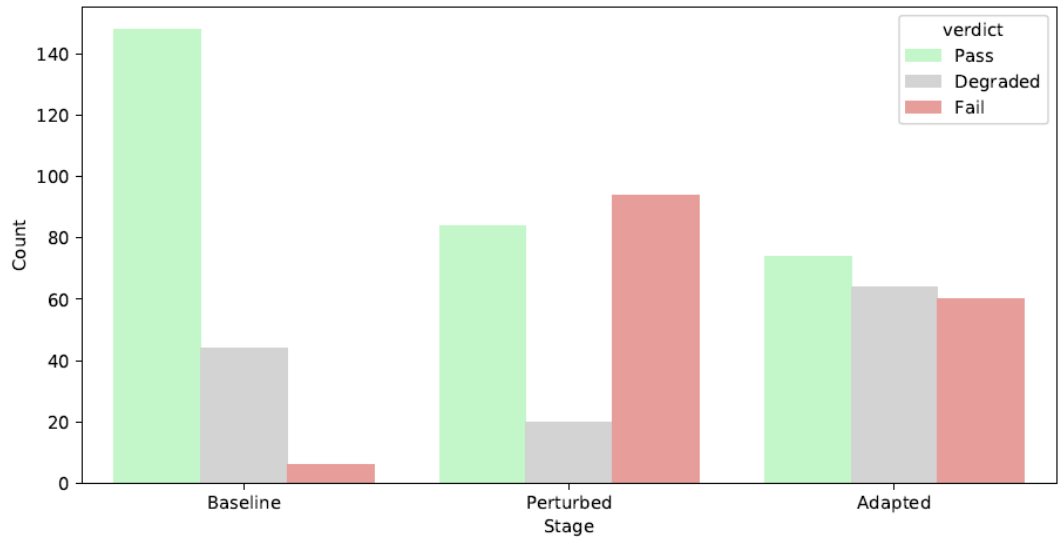

Figure 2: Results of MIT Lincoln Laboratory evaluation of control adaptation and verification approach

Conclusion

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

Lessons Learned

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 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 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.

Secondly, 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 machine learning 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 software components we worked on, which were relatively simple, 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 over-eager 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 where prior knowledge is lacking or non-existent. Overall, both are necessary to provide meaningful and practical adaptations.

Acknowledgement

This material is based upon work supported by the United States Air Force and the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-16-C-0045. The views,

opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

[1] E. Elnahrawy and B. Nath, “Context-aware sensors,” presented at the European Workshop on Wireless Sensor Networks, 2004, pp. 77–93.

[2] R. Tibshirani, “Regression shrinkage and selection via the lasso,” *J. R. Stat. Soc. Ser. B Methodol.*, pp. 267–288, 1996.

[3] N. M. Nasrabadi, “Pattern recognition and machine learning,” *J. Electron. Imaging*, vol. 16, no. 4, p. 049901, 2007.

[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 Conference and Exposition, 2018.

[6] M. Kwiatkowska, G. Norman, and D. Parker, “PRISM 4.0: Verification of Probabilistic Real-Time Systems,” in *SpringerLink*, 2011, pp. 585–591.

[7] B. Lacerda, D. Parker, and N. Hawes, “Multi-objective policy generation for mobile robots under probabilistic time-bounded guarantees,” 2017.