

Massachusetts Institute of Technology
Lincoln Laboratory

**Building Resource Adaptive Software Systems - Phase 2 Evaluation
BAE: Regenerative INtent-Guided Systems (RINGS)**

*Timothy Braje, Robert Cunningham, Jeffrey Hughes, Eric Kilmer,
Timothy Meunier, W. Konrad Vesey, Philip Zucker, Martine Kalke*

September 5, 2018

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

This material is based upon work supported by the Defense Advanced Research Projects Agency under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency.

Lexington

Massachusetts

1. EXECUTIVE SUMMARY

In phase 1 RINGS was developed, which provided a general way to augment existing applications with the goal of making them more resilient to changes within their operating environment. This was achieved through the development of a monitoring architecture which detected functional deviations from intent as specified in the FIDL specification language. When the monitors detect conditions requiring repair, a genetic search algorithm was triggered, guided by a fitness function. If a suitable modification was generated, the application was re-deployed with that change incorporated.

The three challenge problems in phase 2 extend on this previous work to explore both functional and non-functional application intent. In addition, a new approach to developing intent preserving applications via correct-by-construction formal specification is presented. Challenge problem 1 ATIF Auto-Focus demonstrates RINGS genetic algorithms adapting the configuration parameters to produce an improved detection picture of the observed tracks. Challenge problem 2 Legacy Optimization and Repair is a continuation of the phase 1 work to repair message parsing code in the presence of perturbations to the incoming message format and adds a degree of non-functional intent by placing constraints on the amount of system memory the tracker is allowed to utilize. Finally, challenge problem 3 Adapting Formal Specifications focuses on adapting formal specifications to address both functional adaptation and refinement of the implementation to optimize for resource utilization.

1.1 SUMMARY RESULTS AND RECOMMENDATIONS

Each platform and challenge problem is assessed on a number of evaluation criteria which are described in the BRASS Phase 2 Evaluation Methodology [1]. The Phase 2 Evaluation Methodology broadly addresses the general testing strategy as applied to all performers. In the table that follows, we produce a summary of some of the metrics called out in that document.

	CP1	CP2	CP3
Intent Discovery			
Correct	✓	✓	✓-
Complete	✓	✓-	✓-
Specific	✓	✓-	✓-
Fidelity	-	-	-
Test Coverage	small sample	small sample	small sample

TABLE 1

Summary statistics for BAE’s platform. For test coverage, we completely sampled the provided test input files, but these represent a very small fraction of the entire set of possibilities.

This phase, we had more problems getting successful test runs. During the testing period, we were not able to determine whether this was due to environment issues, test harness issues, or just system fragility. It is probably some combination of all of them. We observed successful RINGS repair outcomes for each challenge problem except for the challenge problem 3. We weren't always able to accurately measure the validity of those repairs. We did see some cases where incorrect repairs were being found by the system. More careful tuning of the fitness functions and system monitors could (we believe) help alleviate this problem.

One more general difficulty we had in evaluating the system with the test harness is insufficient verdict measurements. For example, we had no good way of measuring the before and after memory usage for the memory perturbation in the second challenge problem. For the third challenge problem, there was no automated measure of success – if a repair wasn't triggered, we had to assume that the specification was patched. In the last phase of the program, it would be a more efficient use of resources if submission and validation of test-critical performer-provided data was a requirement for entry to testing.

2. DETAILED RESULTS

2.1 ATIF AUTO-FOCUS

The first challenge problem uses the BAE System's All-source Track and ID Fuser (ATIF) from Phase 1. The ATIF tracker takes target detections from a sensor, such as a radar or camera, over time, and identifies objects moving in the field of view.

Over 100 configuration parameters (aka knobs) customize the ATIF tracker's algorithms for different mission conditions. The complexity of the configuration space requires that an ATIF expert be available in the field to maintain the program. Or, the mission selects from a small set of configurations that may not be optimal for the current mission conditions.

The goal of this challenge problem is to demonstrate RINGS' ability to leverage a genetic algorithms approach to adapt the configuration parameters to produce an improved picture of the observed tracks at runtime.

2.1.1 Testing Strategy

For the purpose of evaluation BAE has provided multiple synthetic data streams which represent the same track data under different environmental conditions. For each of these 26 files, we run the following test procedure:

1. Open up stream to prepare system for data input (beginning with initial configuration)
2. Run the (clean) data through the system
3. Close the stream and store tracks
4. Open up stream to prepare system for data input (initial configuration)
5. Run the perturbed data through the system, with adaptations disabled
6. Close the stream and store tracks

7. Open stream with initial config file
8. Run the perturbed data through the system
9. Wait for repair
10. Close stream and store repaired configuration
11. Open up stream with new config params
12. Run perturbed data through system
13. Close stream and store tracks
14. Compare results from steps 3 & 6, and 3 & 13

The comparisons should show that the tracks obtained from the perturbed data (step #6 and #13) are each compared to the baseline results (step #3). The tracks from the adapted system (step #13) should be much closer to the baseline results than the tracks from the un-adapted system (step #6).

2.1.2 Results and Analysis

After running many of the provided input files, we were only able to get a couple to finish the repair. The others were plagued by crashes of either the DAS or the patching system (GenPatcher). Unfortunately, the comparison results for the successful runs were unsuccessful, meaning that the comparator produced an error during the calculation and provided no result. We were able to see, in both runs, that the fitness function before and after repair showed improvement. We can infer this to indicate that the patching did improve the tracks. We have little insight as to why the comparisons were unsuccessful.

2.2 LEGACY OPTIMIZATION AND REPAIR

The second challenge problem consists of two sub-challenge problems to repair legacy C/C++.

CP2a, Message Format Repair, is a continuation of the phase 1 work to repair message parsing code in the presence of perturbations to the incoming message format. Genetic algorithms are again the focus of the repair approach, with updated repair and localization algorithms that are expected to address a wider number of perturbations than phase 1. Repair restores the correct operation and output of the tracker with the new input message format.

CP2b, Resource Usage, is new in this phase and addresses resource utilization. Specifically, the amount of runtime memory available to the track is explored in this phase. Runtime memory can be correlated with the number of concurrent tracks that the tracker has to keep tabs on. The adaptation for this challenge problem focuses on refining the tracker algorithms to optimize for memory. This challenge problem will also rely on the genetic algorithm approach to achieve this.

2.2.1 Testing Strategy

Simulated input track data in the STANAG 4607 format were generated off-line and provided to support the evaluation of each sub-challenge problem. For CP2a, these thirteen files represented perturbations on the file format. For CP2b, the two files had a much larger number of tracks,

triggering an adaptation which aims to decrease the memory usage of the tracker. The testing procedure for these test cases is to:

1. Send unperturbed file
2. Send perturbed file, disabling adaptation
3. Send perturbed file, enabling adaptation
4. Resending the same perturbed file (to check how well the new system handles the perturbed file)
5. Compare results from 1 & 2 and 1 & 4, success being indicated by much better comparisons for the adaptive system

2.2.2 Results and Analysis

For CP2a, we attempted to run through all of the provided input files, but had difficulties getting the system to behave. We saw crashes with GenPatcher (the genetic algorithm sub-system). We also had problems with the solutions that GenPatcher found – producing (presumably) invalid solutions where the comparison results returned were ill-defined. Updating the fitness function with a tighter definition of intent did help prevent these ill-defined repairs. We did have some successful repairs, demonstrating that the methodology does work, but perhaps some effort needs to be placed into ensuring that the monitoring and patching algorithms are made more robust.

For CP2b, we were provided two files, one of them larger than the other. We had no success running the larger one (it takes a long, long time to run through localization), but did have successful completion of the smaller one. We don't have a good way of tracking the amount of memory saved after the repair, but the system did report it as being successful. In the next phase, we suggest the performer produce a means of providing this data to the evaluation team.

2.3 ADAPTING FORMAL SPECIFICATIONS (AFS)

The third challenge problem focuses on the adaptation of a formal specification. This challenge problem is based on ATIF, the All-source Track and ID Fuser from phase 1. The ATIF tracker takes target detections from a sensor, such as a radar or camera, over time, and identifies objects moving in the field of view. For this challenge problem, the core tracking algorithms have been replaced with a formal Coq specification of these algorithms. The specification is refined by Fiat to a Haskell implementation, and integrated with the remaining C++ framework.

A Fiat refinement script acts as a compiler turning the Coq program into a Haskell implementation. The functional adaptation, uses GenPatcher's genetic algorithms to alter the program when an assumption about the environment changes. Environmental assumptions are built into the proofs of the component, and, when these assumptions are violated, the component is allowed to fail with unspecified behavior. RINGS monitors these environmental assumptions and triggers adaptation when a failure is detected.

The assumption is that incoming detections arrive time-ordered. The perturbation will swap detection order such that they do not arrive in order, but are equivalent when sorted. The expected

repair would remove the time-ordered assumption and either: add a filter to drop late arrivals; or add late arrivals to the processing and reprocess from that point.

2.3.1 Testing Strategy

The test is a 4607 data file containing track detections. The file conforms to the expectations/assumptions built into the tracking algorithms’ formal specification. The perturbed version of this file provides the same detections, but reorders them to introduce out of sequence arrivals of detections. The testing procedure is the same as in the previous challenge problem.

2.3.2 Results and Analysis

In running the test, we were unable to trigger an adaptation. We worked with BAE to resolve the problem, but couldn’t determine the reason for the error. We do know that for some reason, the instrumentation messages from the Coq tracker were not reaching the monitors. It is unclear what about our testing environment or deployment caused this difference in behavior from what BAE was able to observe. BAE has been able to use this same system to trigger and repair the application. We refer the reader to BAE’s phase II Platform Workshop presentation.

REFERENCES

- [1] MIT Lincoln Laboratory, “BRASS Phase 2 Evaluation Methodology,” (Sep 20, 2017), URL https://github.mit.edu/brass/mitll/blob/master/documents/evaluation/evaluation_methodology.md.

©2018 Massachusetts Institute of Technology.

Delivered to the U.S. Government with Unlimited Rights, as defined in DFARS Part 252.227-7013 or 7014 (Feb 2014). Notwithstanding any copyright notice, U.S. Government rights in this work are defined by DFARS 252.227-7013 or DFARS 252.227-7014 as detailed above. Use of this work other than as specifically authorized by the U.S. Government may violate any copyrights that exist in this work.

Massachusetts Institute of Technology
Lincoln Laboratory

**Building Resource Adaptive Software Systems - Phase 2 Evaluation
BBN: Interfaces, Models, and Monitoring for Resource-aware Transformations
that Augment the Lifecycle of Systems (IMMoRTALS)**

*Timothy Braje, Robert Cunningham, Jeffrey Hughes, Eric Kilmer,
Timothy Meunier, W. Konrad Vesey, Philip Zucker, Martine Kalke*

August 30, 2018

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

This material is based upon work supported by the Defense Advanced Research Projects Agency under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency.

Lexington

Massachusetts

	CP1	CP2	CP3
Intent Discovery			
Correct	✓	✓	✓
Complete	✓	✓-	✓
Specific	✓	✓	✓
Fidelity	1	1	1
Test Coverage	large sample	complete	complete

TABLE 1

A summary of BRASS results for BBN.

1. EXECUTIVE SUMMARY

The challenge problems provided by BBN Technologies utilized the IMMORTALS program to focus on server-client model scenarios that would be seen during a mission. In challenge problem 1, the IMMORTALS situational awareness application utilizes discovery tools to find where an application is making SQL queries and automatically migrates these queries to conform to a new database schema. In challenge problem 2, applications must be modified to ensure the proper encryption scheme is used to securely transmit data while also preserving the functionality of the system. In challenge problem 3, the IMMORTALS program seeks to demonstrate the automated reasoning about and patching of third-party libraries.

1.1 SUMMARY RESULTS AND RECOMMENDATIONS

Each platform and challenge problem is assessed on a number of evaluation criteria which are described in the BRASS Phase 2 Evaluation Methodology [1]. The Phase 2 Evaluation Methodology broadly addresses the general testing strategy as applied to all performers. In Table 1, we produce a summary of some of the metrics called out in that document.

Overall, the tests went very well. After fixing some network connectivity problems during execution on AWS, to which the team from BBN was very responsive and helpful, all tests ran as intended with results that aligned with BBN’s expectations. The delivery of CP2 was late, but we were eventually able to test it without any problems. In all, it was a successful evaluation! Our recommendation for the next phase of the program would be to expand the number of samples available to use as perturbations for the server and client library upgrades in CP3. More libraries would present a stronger argument that the Immortals system is generic enough to apply to a variety of libraries, and not just the two chosen in this Phase. Furthermore, the lack of a reproducible docker container made risk reductions and the final testing setup more difficult due to the inability to reuse existing techniques and solutions created for all other performers. A deeper look into building a docker environment for Immortals would make batch testing and evaluation easier in the next Phase.

2. DETAILED RESULTS

2.1 DATABASE SCHEMA MIGRATION

Many applications make use of relational databases for storing and retrieving data. The IMMORTALS situational awareness application uses such a database for storing location history, as well as for storage of points of interest and map data. The server also stores locations, but for all the clients. Relational databases have a schema, which describes the layout and content of the data within the database. As applications evolve, and the usage changes, sometimes the original layout of the database can become inefficient for the kinds of queries that are being run: it could be that much more data than was originally intended is being stored, it could be that the database was designed for a slightly different use than what the application is currently doing.

What often happens in these situations is that a human database expert will analyze the performance of the queries against the current data sets and will propose a new database layout, or normalization, that improves the performance of the queries the applications are actually using. Once this layout change is put into place, all the SQL queries that an application makes must be migrated to the new database schema.

The scenarios for this challenge problem address this last issue: migrating the SQL queries to the new database schema. The key technology for this is the CASTOR relational learning algorithm. The approach utilizes discovery tools to find where the application is making SQL queries. Once found, the SQL queries are automatically migrated to conform to the new database schema.

2.1.1 Testing Strategy

Test design for this challenge problem is straightforward. There are 12 different column names with no constraints in how they are arranged and only that they all be included in the new tables provided. Since there are 4,213,597 (the Bell Number [2] for 12) possible set partitions, we only take a small random sampling.

2.1.2 Results and Analysis

When our tests complete, the SUT sends back a done-message that contains a listing of 16 regression tests and their result. After conversing with BBN, we decided that for each complete test case, we plot the percentage of passed regression tests along the x-axis against the test group identifier along the y-axis. After running 28 tests, we can see in Figure 1 that almost all tests perform the same, no matter the configuration of the columns into different tables.

The only outlier (test group ID 368d0d96-f4ed-4ad3-a68f-f9cb68048100) is the first test group where the Adapted system missed passing one more regression test than the others. This regression test is named `testImageSave`, which is not related to the schema adaptation since that test passes in the Perturbed stage also. The fail could be caused by the regression test thread being interrupted or terminated, possibly due to an unexpected exception or another test submission.

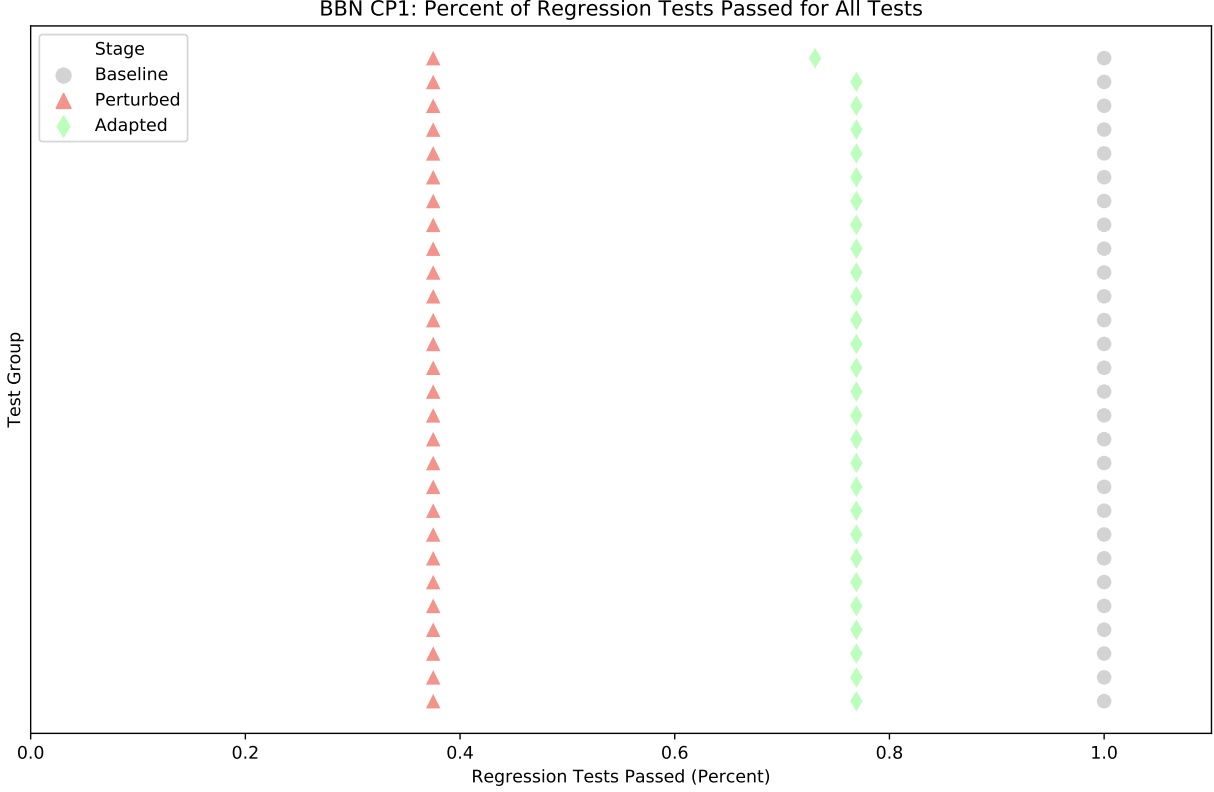


Figure 1. Percent of regression tests passed per test group for CP1.

According to the BBN team, these results are expected:

1. Schema perturbation is not the significant factor for successful query repair in their approach. Underlying data properties of the relation to each query's results is more important (e.g., the relative data cardinalities of the query's projection, filter, etc.).
2. This corroborates theoretical results for the relational learning algorithm they are using, namely, schema independence. Based on several design choices, the relational learning algorithm is able to produce stable results over different schema.
3. The successful regression tests for the Perturbed stage are to unrelated parts of the system that have nothing to do with the queries. The schema perturbations likely break 100% of the queries each time. Hence, there is a bigger leap between the Perturbed and Adapted stages if we exclude these unrelated tests.
4. The reason the Adapted stage does not achieve 100% regression test pass is due to the inability of the relational learning algorithm to completely repair queries that handle different data types.

Looking at the verdict plots in Figure 2, we can see that all tests completed and resulted in a Degraded verdict as explained by statement number 4 above.

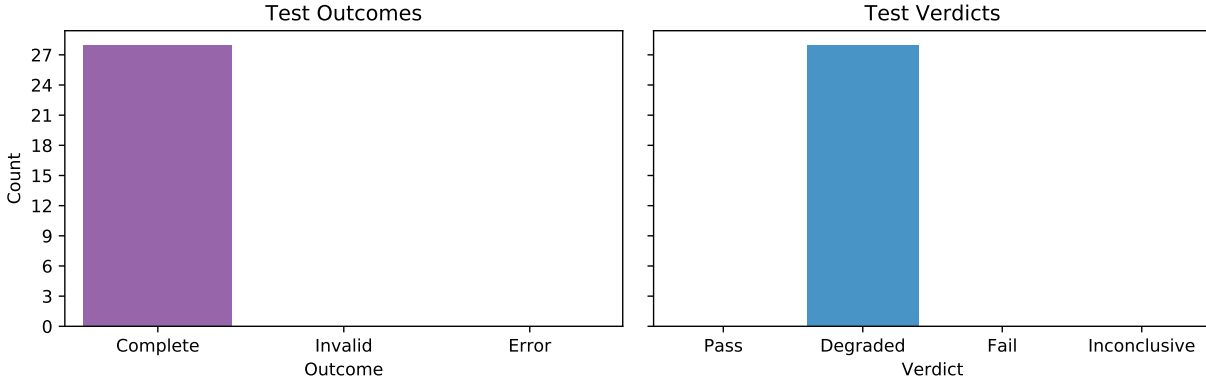


Figure 2. Verdicts and outcomes for CP1.

2.2 CROSS-APPLICATION DEPENDENCIES

The second challenge problem investigates the change of drivers in the middleware space. This will be demonstrated using the client-server interaction in the IMMORTALS tactical situational awareness application. The ATAK Lite clients use a socket-based middleware that uses COT messages to periodically transmit situational awareness information (such as location) to the MARTI server.

Suppose a new requirement is asserted specifying that all messages traversing networks must be encrypted. It is not enough to simply add encryption logic on the client side. The server must also be adapted to properly decrypt the information before using it. The goal of this challenge problem is to demonstrate the capability of addressing changes in a way which preserves the functionality of the systems by adapting to changing mission requirements.

2.2.1 Testing Strategy

Test design for this challenge problem is straightforward. There are 4 different values for the security standard that can perturb the system. We completely test all 4.

2.2.2 Results and Analysis

Similar to CP1, the SUT for CP2 sends back a done-message that contains a listing of 18 regression tests and their result. Again, we plot the percentage of passed regression tests along the x-axis against the new cryptography security standard perturbation along the y-axis. After running through all of the different cryptography security standards, we can see in Figure 3 that the tests follow exactly what we would expect: The Adapted and Baseline stages pass all regression tests, and the Perturbed stage fails regression tests related to the secure data transmission.

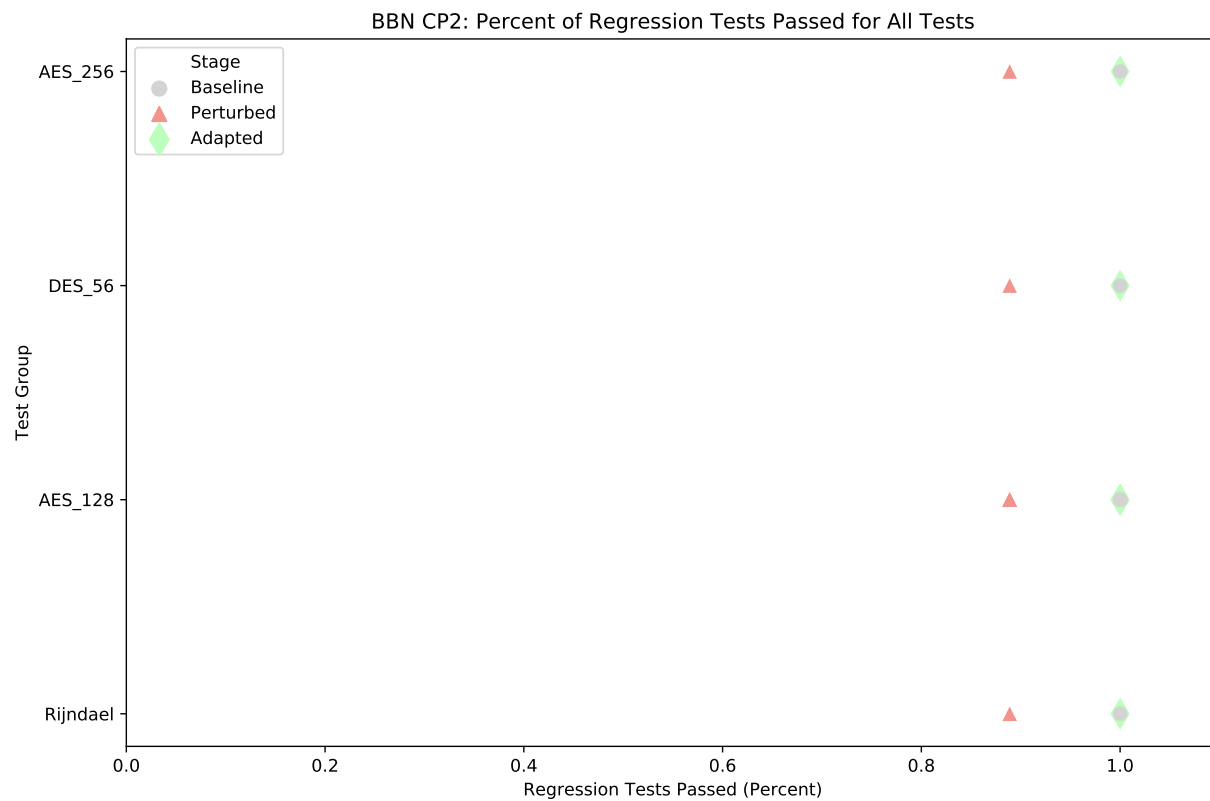


Figure 3. Percent of regression tests passed per test group for CP2.

Looking at the verdict plots in Figure 4, we can see that all tests completed and resulted in a Pass verdict!

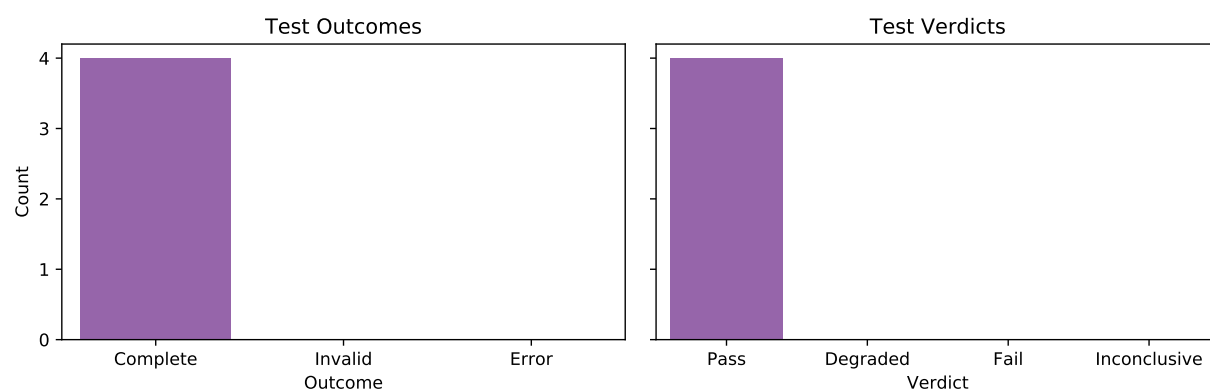


Figure 4. Verdicts and outcomes for CP2.

2.3 THIRD-PARTY LIBRARY DEPENDENCY UPGRADE

Modern software is built upon many layers of third-party libraries and operating system features. These third-party libraries are constantly updated and upgraded with new features and security patches. While security patches are time-sensitive, there is typically no guarantee that a library vendor will invest the time and energy to backport fixes to the version that an application currently depends on. Additionally, upgrading the system to use the newest version of a library can break the dependent application. The goal of this challenge problem is to demonstrate the automated reasoning and patching of third-party libraries. The DAS provides a way to pick combinations of changes, and then fills out the rest of the change request, metadata information regarding the requested change and mitigation taken, and possibly generating a new version of the library.

2.3.1 Testing Strategy

Test design for this challenge problem is straightforward. There are 3 different values to perturb the system and patch different libraries. We test all 3 values.

2.3.2 Results and Analysis

Similar to CP1 and CP2, the SUT for CP3 sends back a done-message that contains a listing of 16 regression tests and their result. Again, we plot the percentage of passed regression tests along the x-axis against the new third-party library dependency upgrade perturbation along the y-axis. After running through all of the different third-party library dependency upgrade perturbations, we can see in Figure 5 that the tests follow exactly what we would expect: The Adapted and Baseline stages pass all regression tests, and the Perturbed stage fails regression tests related to the upgrades.

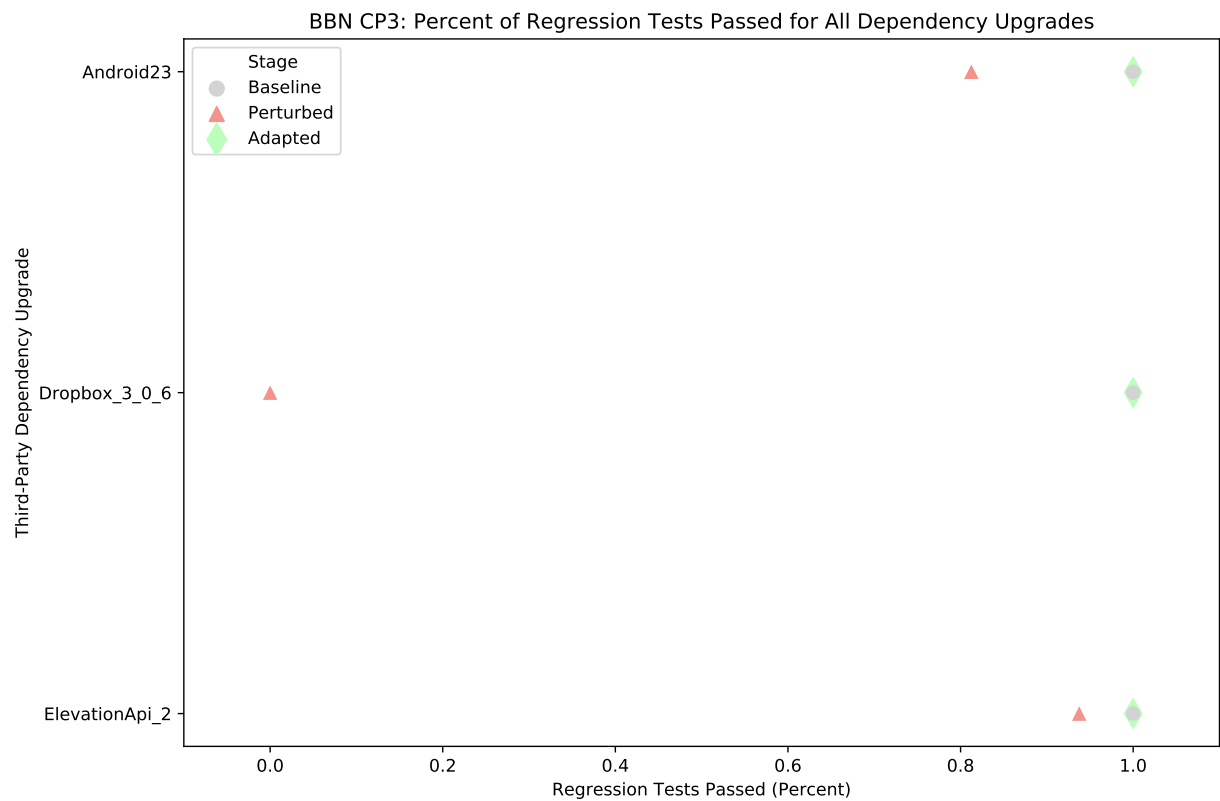


Figure 5. Percent of regression tests passed per test group for CP3.

Looking at the verdict plots in Figure 6, we can see that all tests completed and resulted in a Pass verdict!

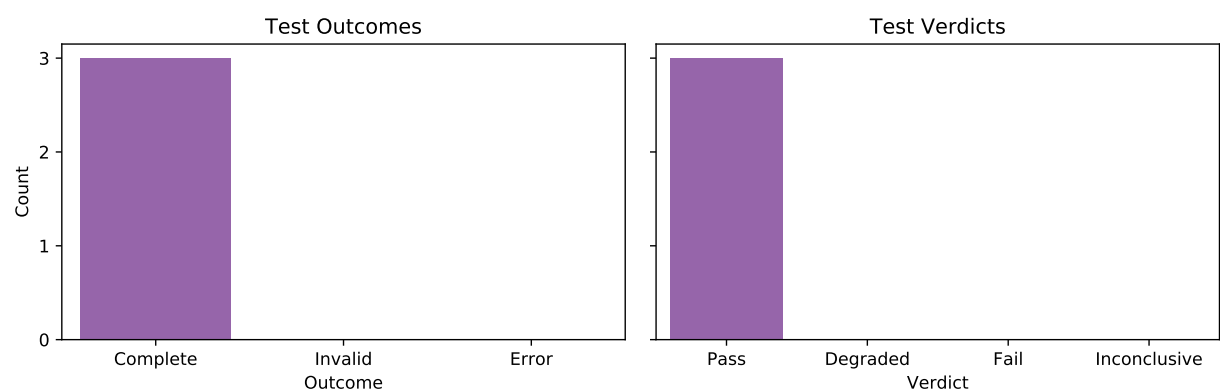


Figure 6. Verdicts and outcomes for CP3.

REFERENCES

- [1] MIT Lincoln Laboratory, “BRASS Phase 2 Evaluation Methodology,” (Sep 20, 2017), URL https://github.mit.edu/brass/mitll/blob/master/documents/evaluation/evaluation_methodology.md.
- [2] “Bell Number,” https://en.wikipedia.org/wiki/Bell_number, accessed: 2018-08-17.

©2018 Massachusetts Institute of Technology.

Delivered to the U.S. Government with Unlimited Rights, as defined in DFARS Part 252.227-7013 or 7014 (Feb 2014). Notwithstanding any copyright notice, U.S. Government rights in this work are defined by DFARS 252.227-7013 or DFARS 252.227-7014 as detailed above. Use of this work other than as specifically authorized by the U.S. Government may violate any copyrights that exist in this work.

Massachusetts Institute of Technology
Lincoln Laboratory

**Building Resource Adaptive Software Systems - Phase 2 Evaluation
CMU: Model-based Adaptation for Robotics Systems (MARS)**

*Timothy Braje, Robert Cunningham, Jeffrey Hughes, Eric Kilmer,
Timothy Meunier, W. Konrad Vesey, Philip Zucker, Martine Kalke*

September 4, 2018

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

This material is based upon work supported by the Defense Advanced Research Projects Agency under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency.

Lexington

Massachusetts

1. EXECUTIVE SUMMARY

In Phase 1, the MARS team explored the domain of autonomous robotics with the aim of improving autonomous navigation and automatic sensor-calibration for systems encountering unplanned environmental events. In addition, challenge problem 1 explored the robot’s newfound ability to consider its energy consumption and remaining battery when making runtime route planning decisions. The success of each challenge problem was based on the robot’s ability to navigate within its environment to a pre-defined target location. The three challenge problems proposed in this Phase continue to utilize a similar mission context for measuring success.

In this Phase, the first challenge problem aims to demonstrate advances in the robot’s capability to learn power models efficiently under budget constraints. The goal of this challenge problem is to discover the power model of the mobile robotics platform and use the discovered model to adapt for optimal mission performance. The challenge problem builds on techniques developed in Phase 1, which incorporated measurements of energy consumption and remaining battery charge when making runtime route planning decisions, and extends this work by adding an off-line learning component. The intent is to provide greater resiliency to changes in available hardware, algorithms and workload while also dealing with potential environmental changes such as obstacles in the planned route. Similar to Phase 1, the mission objective for this challenge problem requires the robot to navigate to one or more target locations.

The second challenge problem is new for Phase 2 and focuses on demonstrating code-level adaptation in response to source-code perturbations representative of commonly encountered bugs in robotics software. The challenge problem aims to automatically identify and repair bugs that are exposed (or introduced) during the natural evolution of robotics systems. To tackle this problem, a search-based program repair technique was developed to efficiently find and fix multiple classes of bugs. The system was evaluated by subjecting it to realistic faults (referred to as perturbation scenarios) and measuring the degree to which intent (represented by a fixed set of system-level tests) was recovered. A number of the system-level tests required the robot to successfully navigate within its environment.

The third challenge problem is an extension of the robotic navigation and sensor challenge problem explored in Phase 1. The goal of the challenge problem is to demonstrate advances in MARS’ capability to adapt to perturbations such as software component failure and/or environmental change. Relational and probabilistic multi-objective planning were leveraged to consider a range of possible adaptations including software configuration, software component parameters and mission level adaptations. As compared to Phase 1 where perturbations predominately triggered mission changes, the perturbations available in this Phase were designed to also trigger software configuration changes that are to be handled on-line (i.e. changes to actual source code). The mission for this challenge problem remains the same, to navigate to a pre-defined target location.

1.1 SUMMARY RESULTS AND RECOMMENDATIONS

Each platform and challenge problem is assessed on a number of evaluation criteria which are described in the BRASS Phase 2 Evaluation Methodology [1]. The Phase 2 Evaluation Methodology

	CP1	CP2	CP3
Intent Discovery			
Correct	✓	✓	✓
Complete	✓	✓	✓
Specific	✓	✓	✓
Fidelity	0.94	0.87	0.89
Test Coverage	small sample	small sample	small sample

TABLE 1

Summary statistics for CMU MARS platform.

broadly addresses the general testing strategy as applied to all performers. In the table that follows, we produce a summary of some of the metrics called out in that document.

Overall, the tests went very well. There were some difficulties on some of the test cases where it was difficult to find valid test cases or to produce tests that didn’t cause the system under test to fail. The CMU Mars system provided a rich API for the evaluators to exercise and explore the robustness of the system. During testing, we did uncover some unexpected bugs and corner cases. In all it was a successful evaluation! Our recommendation for the next phase of the program would be to make some effort to uncover reasons why some of the test scenarios seemed to imply fragility of the system under test. Maybe it is problems with the underlying simulation framework, perhaps it has something to do with the design/construction/implementation of the test cases, or maybe it points to improvements that could be made to the technology being developed. Though the API supported more complex test cases, the only ones that were executed were ones that involved single kinds of perturbations. In the next phase, more may be learned by trying out combinations of perturbations and seeing how they interact with one another.

2. DETAILED RESULTS

2.1 INTEGRATED POWER MODEL DISCOVERY AND ADAPTATION

The aim of the first challenge problem is to apply learning techniques to discover new power models for a robotics system, and take advantage of these new models at runtime to improve the quality of missions of the robot over the use of no or static models. This challenge problem leverages techniques developed in Phase 1, which incorporate measurements of energy consumption and remaining battery charge when making runtime route planning decisions, and extends this work by adding an off-line learning component. The intent is to simulate robots with different hardware, algorithms, and workload and therefore different power usage characteristics. The challenge problem intentionally simulates severe changes that might be caused by unanticipated and yet-unknown future environment changes in order to test the degree to which the system is able to support a higher-level navigation mission scenario by learning and adapting to these changes.

2.1.1 Testing Strategy

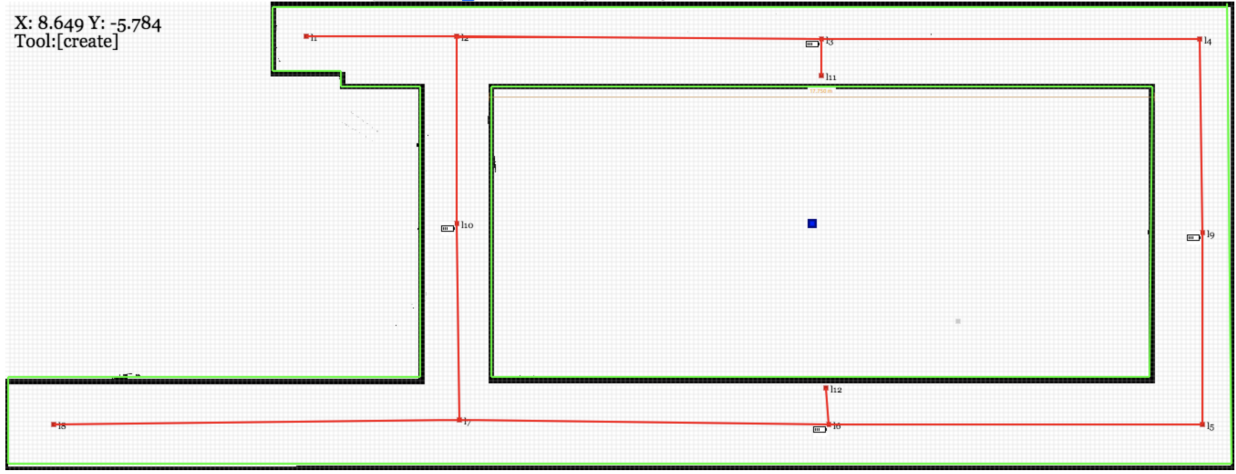


Figure 1. Map used for challenge problem 1.

For this challenge problem, the map (Figure 1) is kept static. The tests are broken up into two different categories: battery perturbations and obstacle perturbations. For both of these tests, we generate a list of tasks (path that the robot needs to traverse) for the robot to accomplish by randomly selecting 1) a number of way points to generate and 2) the actual list of way points. Given the path, we randomly choose:

1. The power model
2. The ‘discharge budget’ (maximum number of queries allowed against the discharge function during learning)
3. Test category (battery perturb or obstacle placement)

For the battery perturbation cases, we randomly select a number of times (1-5) to discharge the battery. For each of the discharge events we randomly select: the amount of discharge (1-10%) and the wait time from the previous discharge event (30-300 seconds). These values represent a static discharge schedule that is applied dynamically during test execution. For obstacle perturbation tests, we randomly choose a place in the path to place the obstacle (between two of the way points from the generated task list).

The metric for determining success is to take the fractional number of way points that the robot actually passed through (to within a certain amount) during execution. Thus, if a robot goes through all way points, it receives a score of 1, while if it goes through none it receives a score of 0. It can also obtain a zero score if it either:

1. Doesn’t go through the same number of way points as it was supposed to
2. Goes through the way points in the wrong order

As we discuss in the results, the first point arises repeatedly in the battery drain tests.

2.1.2 Results and Analysis

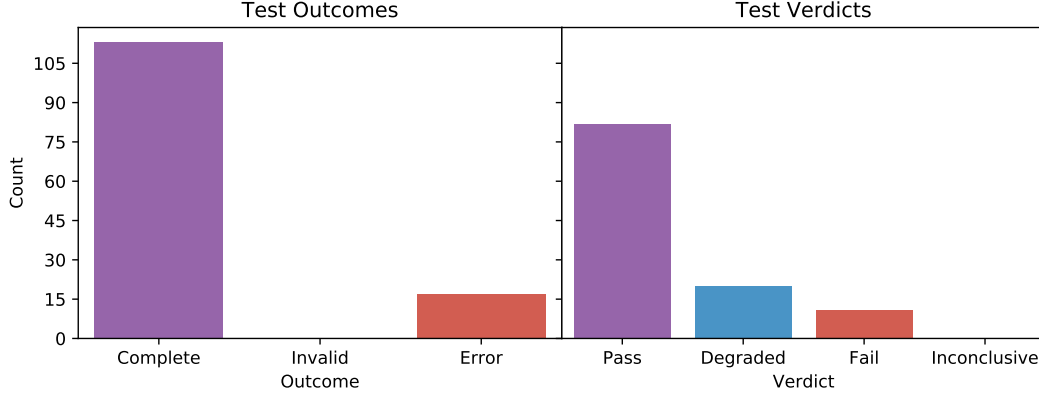


Figure 2. Verdicts and outcomes for the obstacle test cases.

Discussing the obstacle test cases first, we see (in Figure 2) a high percentage of the test cases run produced ‘complete’ outcomes. We were able to achieve this because of the way we constructed the tests. In selecting adjacent way points, we were careful to choose in such a way that when we placed obstacles between way points, we were highly confident that the robot would choose the path with the obstruction. Obstacles were also placed in such a way that we wouldn’t construct test cases where the robot couldn’t achieve a task (i.e. that there are always alternative paths to go around the obstacle and get to the desired way point). We can see by looking at the verdicts that the adapted system tends to recover functionality, only producing failures in about 20% of the test cases, most of which are due to the way the success metric is computed (as discussed above).

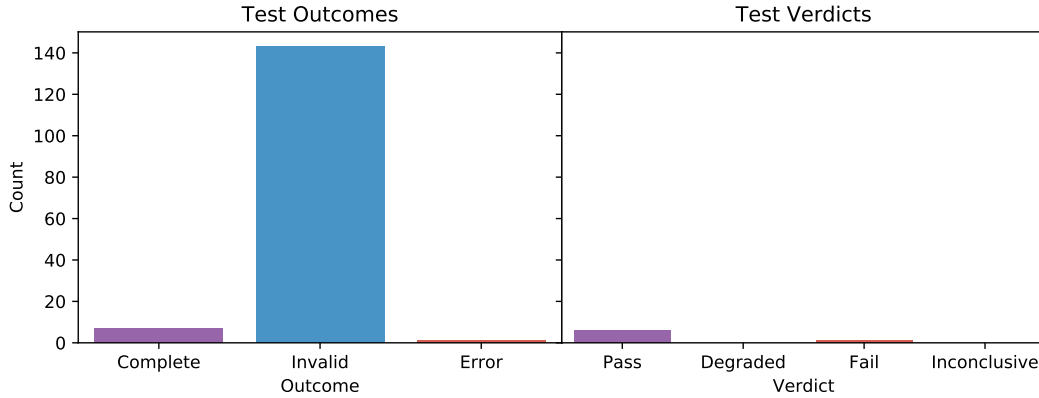


Figure 3. Verdicts and outcomes for the battery test cases.

Generating valid test cases for the battery tests turned out to be a bit more challenging. We see a high number of ‘invalid’ outcomes (Figure 3). The reasons for this are clear – the only way a test can fail is if the robot runs out of battery. The only way a robot can run out of battery is if it gets drained during traversal of the path, or if we perturb the battery and drain it more quickly. Adding too many battery drain perturbation events will result in a test case where even the adaptive system can’t recover. Having too few results in test that are too ‘easy’, so the Baseline B stage succeeds. We found a very large number of tests in this category where the perturbation drains the battery completely. These tests were marked as invalid. For the tests that do complete, we find that most pass, and that a few fail. In the next phase of the program, the performer and evaluation teams should work closely to design tests that can be randomly generated but which will also exercise the desired parameter space in an interesting way.

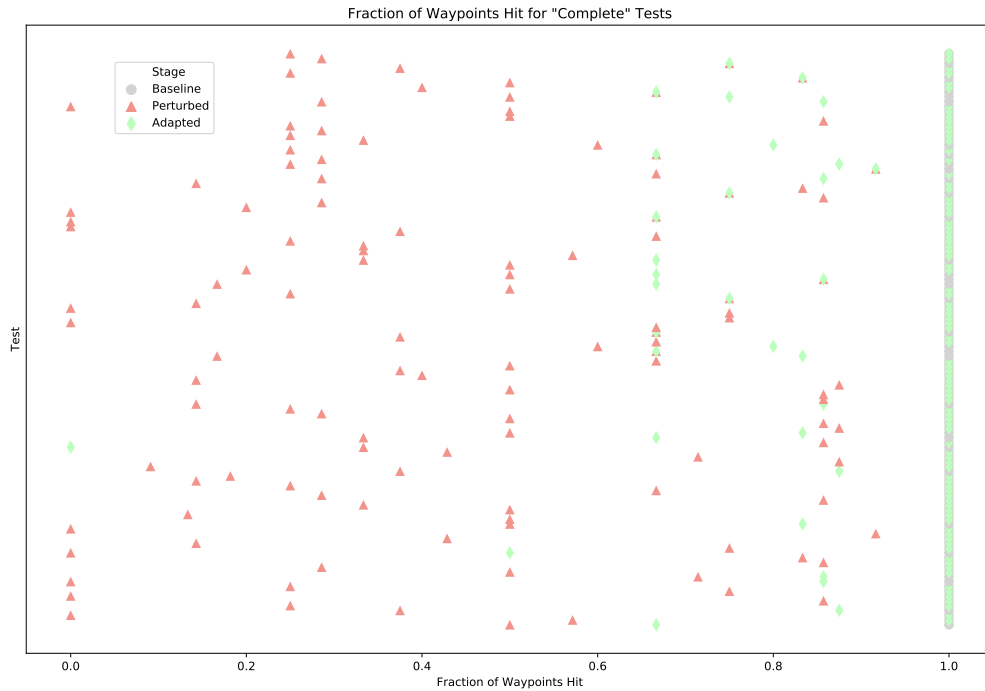


Figure 4. Plot of task recovery as a function of test stage.

To look at the data from another angle, we can plot (Figure 4) the test identifier against the task completion ‘score’ metric. The score metric roughly represents the fraction of the way points which the robot hit during the test execution. We plot this metric for completed tests, finding no surprises in the results. For the Baseline A stage, the robot successfully achieves all tasks (a score of 1.0). For the Baseline B (perturbed) stage, we see a wide variety of scores, representing

reductions in the number of tasks being achieved. For the challenge (adapted) stage, the scores tend to improve.

2.2 CODE-LEVEL ADAPTATION

The second challenge problem demonstrates code-level adaptation in response to source-code perturbations representative of commonly encountered bugs in robotics software. The challenge problem aims to simulate bugs that are exposed (or introduced) during the natural evolution of robotics systems. To tackle this problem, a search-based program repair technique was developed to efficiently fix these bugs rather than attempt to address all theoretically possible bugs. The system is evaluated by subjecting it to realistic faults (referred to as perturbation scenarios) and measuring the degree to which intent (represented by a fixed set of system-level tests) is recovered.

2.2.1 Testing Strategy

Test design for this challenge problem is fairly straightforward. We first randomly select the bug type that we would like to introduce into the system from a list of seven kinds, ranging from deleting void function calls to flipping arithmetic or boolean operators. Once we have chosen the kind of fault to introduce, we select the file we would like to try and introduce the fault into. To do this, we ask the system under test for a list of all source files from which we randomly choose one. After we have a file, we ask the system for a list of possible perturbations that could be applied to the given file to produce a fault of the kind we seek. We iterate over this list of perturbations in a randomly chosen order attempting them in turn. Only perturbations which cause a failure in the test suite are considered valid, since the test suite is what provides the test with a measure of intent to be preserved. If none of the perturbations are deemed valid, we move onto another file and repeat the process until a valid perturbation is found.

2.2.2 Results and Analysis

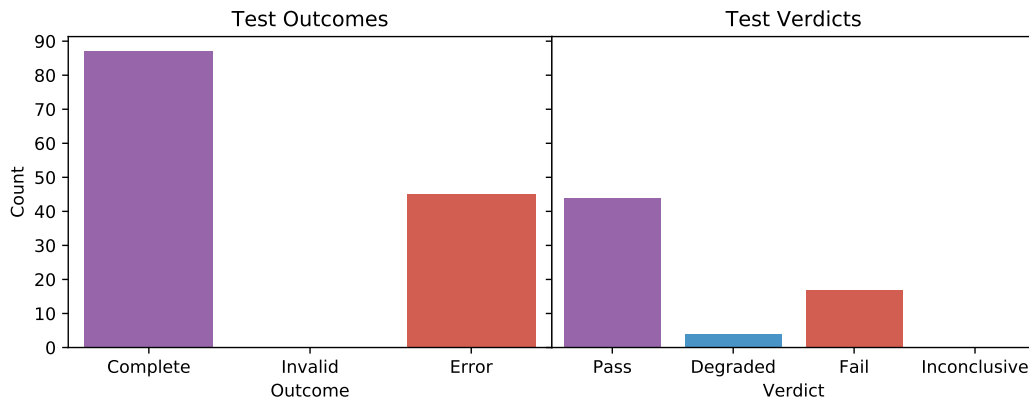


Figure 5. Verdicts and outcomes for code repair.

Looking at the outcome plot in Figure 5, we see that we encounter a fair amount of errors during execution. This appears to be due to the test taking too long to find a valid perturbation to apply. Since the search space is extremely large (the list of files is the entirety of the c code in the Robot Operating System (ROS)) and we are randomly searching for valid perturbations within this rather large search space, we often loop for a while trying to find valid test cases. When this looping goes on too long, one of the systems can crash due to a variety of issues (one thing we noticed was that we had to increase the allowed open file handles within one of the docker containers to be fairly large or the tests tended to error out after only a few attempts at finding perturbations). The search problem is exacerbated by the fact that the applied perturbation needs to cause a test failure in the test suite that comes with ROS. When we do have a valid test case, we find that the system under test does a reasonable job of finding fixes. It succeeds about three times as often as it fails, and only produces degraded results a negligible amount of the time. Here, success is determined by full restoration of the test suite, degradation is partial restoration of the test suite, and failure happens when no tests were able to be corrected.

2.3 ON-LINE ROBOTIC SOFTWARE ARCHITECTURE ADAPTATION TO SOFTWARE FAILURE, UNEXPECTED ENVIRONMENTS

The third challenge problem is an extension of the robotic navigation and sensor challenge problem explored in Phase 1. The goal of the challenge problem is to demonstrate advances in MARS's capability to adapt to perturbations such as software component failure and/or environmental change. Relational and probabilistic multi-objective planning were leveraged to consider a range of possible adaptations including:

- software configuration
- software component parameters
- mission level adaptations.

The mission for this challenge problem is defined by the ability of the robot to successfully navigate through a changing environment in order to reach a pre-defined target location. As compared to Phase 1 where perturbations predominately triggered mission changes, the perturbations available in this Phase were designed to also trigger software configuration changes that are to be handled on-line (i.e. changes to actual source code).

2.3.1 Testing Strategy

Like the first problem, the map (Figure 6) for this one remains fixed for all of the tests. This map is quite a bit larger than the one in challenge problem 1. There are pre-defined dark regions and unsafe areas (containing obstacles). Lights are also spread throughout the map. The goal of the robot is to traverse from a starting point to an ending point, optimizing for different utility functions (timeliness, safety, or efficiency). For this challenge problem, we have three different kinds of tests, where we introduce a sensor failure, a software failure, or turn off lights within the map. For each of these tests, we choose at random:

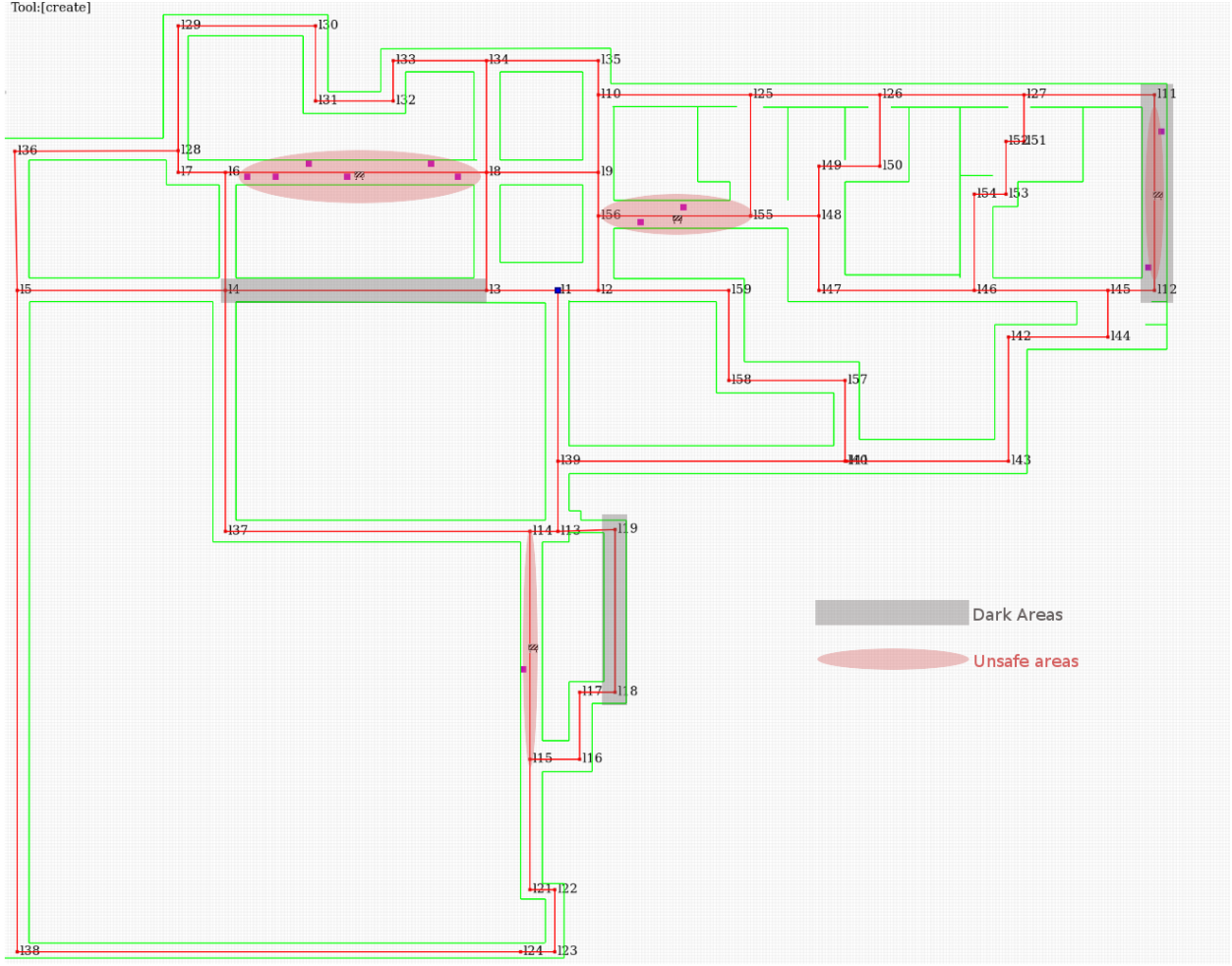


Figure 6. Map used for challenge problem 3.

1. Starting and ending location
2. Utility function to maximize
3. Starting software / hardware configuration (navigation sensors and algorithms)

When we choose the test type (sensor fail, software fail, or decreased lighting conditions), the actual perturbation to perform is based on the robot's initial configuration. The possible starting configurations are:

- amcl-kinect
- amcl-lidar
- mrpt-kinect
- mrpt-lidar
- aruco-camera

When we fail a sensor, we look at the starting configuration and fail the one that is being used (e.g., in the amcl-kinect starting configuration, we would fail the kinect sensor). Since the aruco-camera configuration is the only one sensitive to lighting conditions, when we are in the lighting perturbation test case, we ensure that the starting configuration is aruco-camera.

2.3.2 Results and Analysis

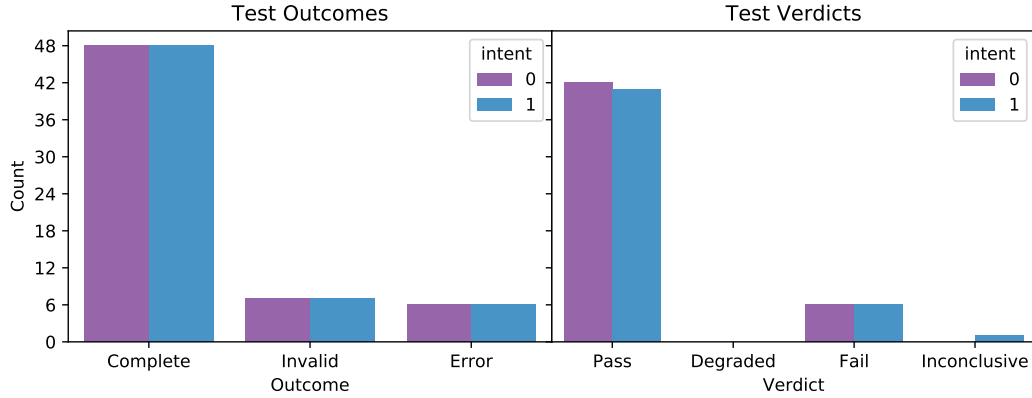


Figure 7. Verdicts and outcomes for the hardware failure test cases

For all three test cases, there are verdicts for each of two intents. Intent 0 measures whether the robot achieved its mission of getting to the final destination. Intent 1 is a measure of how well the robot did in maximizing the desired utility function. In all three test sets, we determine the outcome of the test based on whether intent element 0 was achieved, since that is the primary measure of the robot’s mission. Because of this, you see in the outcome graphs that the results are the same for the two intent elements.

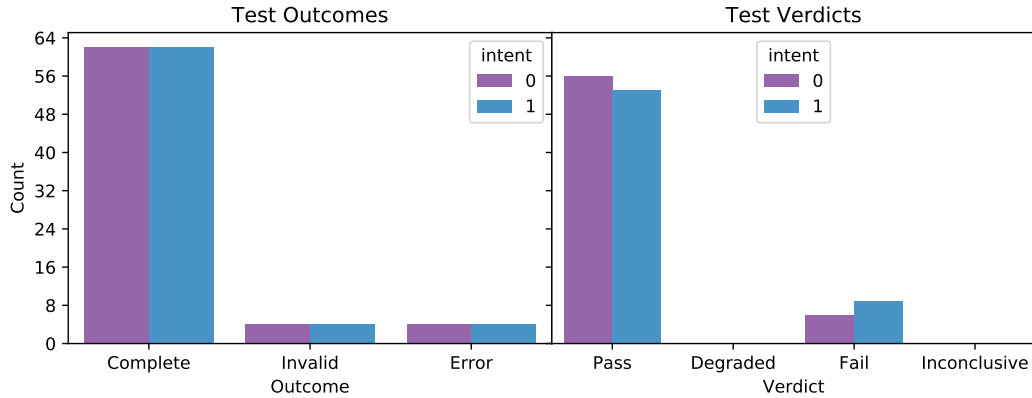


Figure 8. Verdicts and outcomes for the software failure test cases

Looking at the verdict plots for the hardware failure test cases (Figure 7), we see that most generated test cases produced complete outcomes, with only a few producing either invalid test cases or errors. Of the complete test cases, the vast majority pass both intent elements. Interestingly, we see no degraded results, with a few failures. This seems to indicate that the robot is typically able to recover from a sensor failure by choosing a new sensor and updating the navigation algorithm. The robot doesn’t ever seem to just “not get quite close enough to the target.”

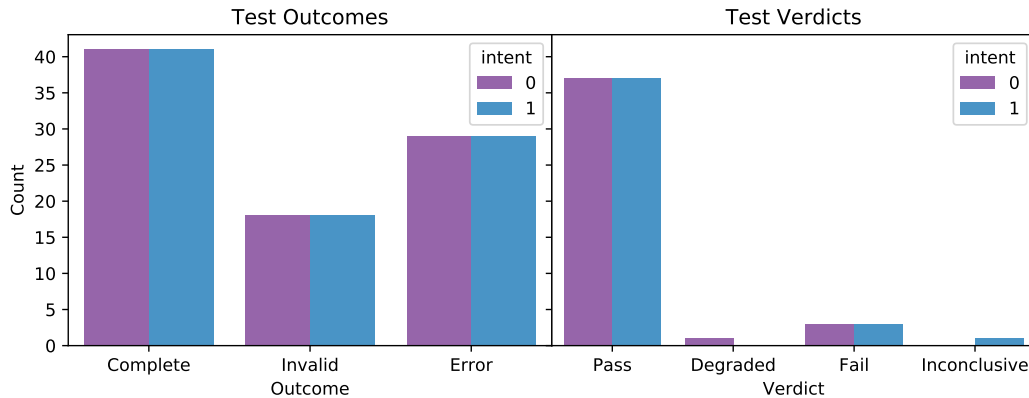


Figure 9. Verdicts and outcomes for the lighting conditions test cases

The results of the software failure test case (Figure 8) are nearly identical to those of the hardware failure test case. We can speculate here that the MARS system’s adaptive capability is largely independent of whether we are looking at a hardware or a software failure, recovering intent for either with roughly the same probability.

The last set of verdict plots (Figure 9) is a little bit different than the previous two. While results in the case of complete tests are roughly the same as the previous test cases, the outcomes have a much wider variation. We believe this to be an artifact of test construction. The way we determine which lights to turn off is entirely based on probabilities. We choose, randomly, what numerical fraction of the lights to turn off, and then randomly select that many lights from the list of lights turned on. In the unlucky case where we don’t turn off lights that would impact the path the robot is going to take, then the result of the Baseline B test would be successful, resulting in an invalid test case. It also appears that messing with the lighting configurations de-stabilizes the simulator somewhat, resulting in many more errors.

We can look at the data in another way. In Figure 10, for each complete test case, we plot the final distance from the target along the x axis against the test identifier along the y axis. For each test, we can see the results for the different stages. From this plot, it becomes apparent that though the Baseline B fails with the robot in a wide range of distances from the target, the adapted system tends to allow the robot to achieve the goal.

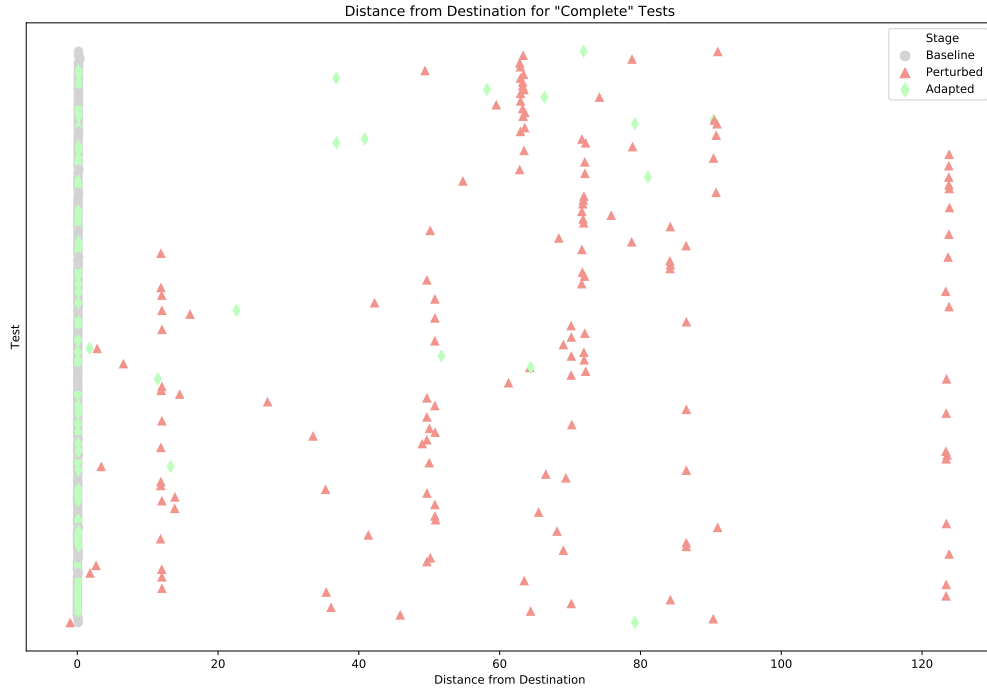


Figure 10. Plot of test versus distance from end goal.

REFERENCES

- [1] MIT Lincoln Laboratory, “BRASS Phase 2 Evaluation Methodology,” (Sep 20, 2017), URL https://github.mit.edu/brass/mitll/blob/master/documents/evaluation/evaluation_methodology.md.

©2018 Massachusetts Institute of Technology.

Delivered to the U.S. Government with Unlimited Rights, as defined in DFARS Part 252.227-7013 or 7014 (Feb 2014). Notwithstanding any copyright notice, U.S. Government rights in this work are defined by DFARS 252.227-7013 or DFARS 252.227-7014 as detailed above. Use of this work other than as specifically authorized by the U.S. Government may violate any copyrights that exist in this work.

Massachusetts Institute of Technology
Lincoln Laboratory

**Building Resource Adaptive Software Systems - Phase 2 Evaluation
CMU SPIRAL**

*Timothy Braje, Robert Cunningham, Jeffrey Hughes, Eric Kilmer,
Timothy Meunier, W. Konrad Vesey, Philip Zucker, Martine Kalke*

September 4, 2018

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

This material is based upon work supported by the Defense Advanced Research Projects Agency under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency.

Lexington

Massachusetts

1. EXECUTIVE SUMMARY

The Phase 1 challenge problems highlighted the adaptability of the system by demonstrating SPIRAL's capability to 1) switch to appropriate routines optimized for the particular system/mission scenario at runtime and 2) switch between two different processor architectures at run-time. In Phase 2, challenge problem 1 extends the first capability by focusing on the online code generation mechanism. The goal of the online code generation mechanism is to reduce the amount of code that has to be generated prior to the start of the mission. Specifically, CP1 intends to demonstrate this capability by reducing the size of pre-generated routines by a factor of 10, while maintaining the performance of the system within 10% of the performance obtained with the large pre-generated library.

Challenge problem 2 extends the second capability developed in Phase 1 to extend this capability to adapt to many more systems, including legacy (end-of-life) systems. The goal of this challenge problem is to demonstrate that the scalability of the code generation software to the number of supported architectures. Specifically, CP2 intends to demonstrate that the code generation system supports systems that have different ISAs, different hardware characteristics (e.g. in-order execution vs out-of-order execution, different cache sizes and associativities), and different number of cores.

Challenge problem 3 takes the SPIRAL team in a new direction for this Phase. Though the team is not addressing the challenge of adaptive systems in the same way as the other performers, they have extended the capability of the system to detect and suggest appropriate changes in order to maintain the performance of the system. Specifically, CP3 intends to demonstrate this capability by detecting that the performance (e.g. runtime) of the system is not meeting the deadline specified, and suggesting appropriate changes to the user.

1.1 SUMMARY RESULTS AND RECOMMENDATIONS

Each platform and challenge problem is assessed on a number of evaluation criteria which are described in the BRASS Phase 2 Evaluation Methodology [1]. The Phase 2 Evaluation Methodology broadly addresses the general testing strategy as applied to all performers. In the table that follows, we produce a summary of some of the metrics called out in that document.

Summary Results

CMU Spiral appears to have achieved what they set out to achieve. In CP1, the timing performance was maintained with a reduction in memory usage. In CP2, the system takes advantage of the extra capabilities offered by the system to increase performance. In CP3, suggestions made upon the event of changing load were able to maintain the performance of the system.

	CP1	CP2	CP3
Intent Discovery			
Correct	✓	✓	✓
Complete	✓	✓	✓
Specific	✓	✓	✓
Fidelity	1	1	1
Test Coverage	complete	medium sample	medium sample

TABLE 1

Summary statistics for CMU SPIRAL platform.

2. DETAILED RESULTS

2.1 PERFORMANCE PER LINE OF CODE

The first challenge problem builds on the work of Phase 1 which demonstrated the ability to adapt applications at runtime by performing a switch to routines which were optimized for the particular system/mission requirement. This, however, required a large static library of routines to be pre-generated prior to the commencement of the mission.

Development in Phase 2 focused on extending the capability of the system to support online code generation via dynamically swapping finer grained library components. The goal of the online code generation mechanism is to reduce the amount of code that has to be generated prior to the start of the mission. Specifically, this challenge problem aims to demonstrate this capability by reducing the size of pre-generated routines by a factor of 10, while maintaining the performance of the system within 10% of the performance obtained with the large pre-generated library. This section defines the parameter space, evaluation criteria, as well as the approach and results of the evaluation for this Phase.

2.1.1 Testing Strategy

For CP1, the system under evaluation is fixed to a specific architecture. The same sweep of requested architecture changes is performed both on a Phase 1 codebase and on the new Phase 2 codebase. The performance results obtained with the Phase 2 system should be within 10% of the performance of the system used in Phase 1.

2.1.2 Results and Analysis

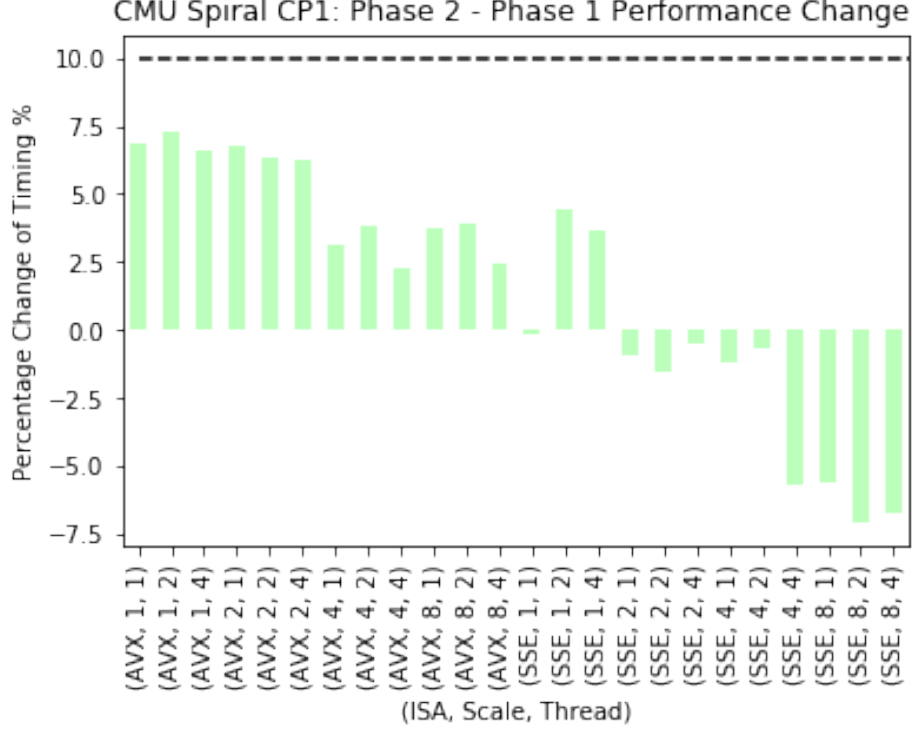


Figure 1

Figure 1 shows a comparison of performance from Phase 1 to Phase 2. The system maintains performance to within 10% while reducing memory usage, achieving the performance goals.

2.2 PORTABILITY TO LEGACY SYSTEMS

The challenge problems in Phase 1 demonstrated the capability of SPIRAL to switch between two different architectures at runtime. In Phase 2 this capability was extended to adapt to many more systems, including legacy systems. The goal of this challenge problem is to demonstrate the scalability of the code generation software by targeting multiple architectures which have different instruction set architectures, different characteristics (e.g. in-order execution vs. out-of-order execution) and different amounts of resources (e.g. number of cores and cache size).

2.2.1 Testing Strategy

For this challenge problem, the architecture adaptability of the system is tested. The performance obtained using the single-thread implementation optimized for the oldest machine architecture available will be used to generate a baseline comparison. This baseline test routine will run

on all available architectures to generate a performance baseline measurement for comparison. The code generator will then be used to generate optimized implementations for each of the different architectures. The performance measurement of this optimized implementation will be compared against the baseline performance measurement. The intent of the system is deemed to be the sustained performance of the adapted code. The performance of the system with the generated code should be higher than the baseline implementation for each architecture.

2.2.2 Results and Analysis

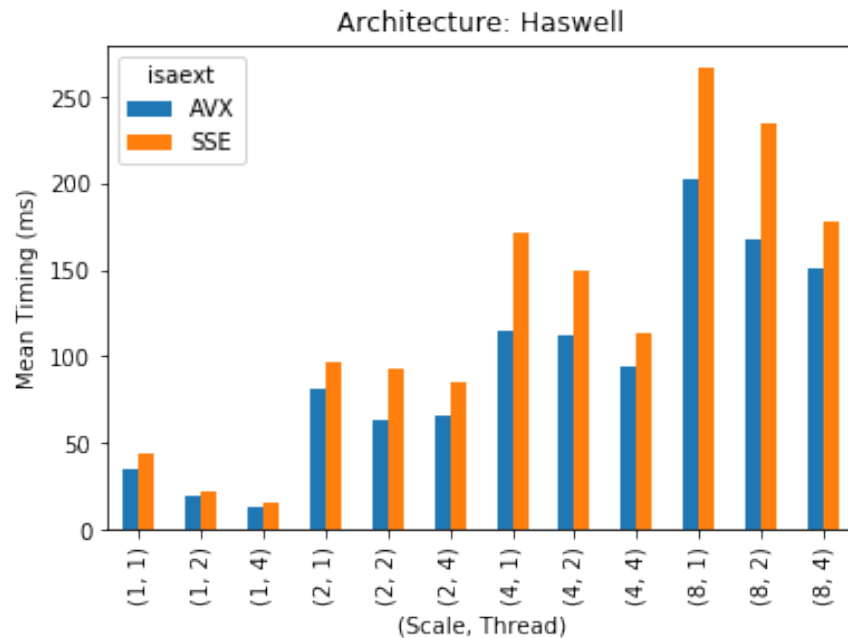


Figure 2

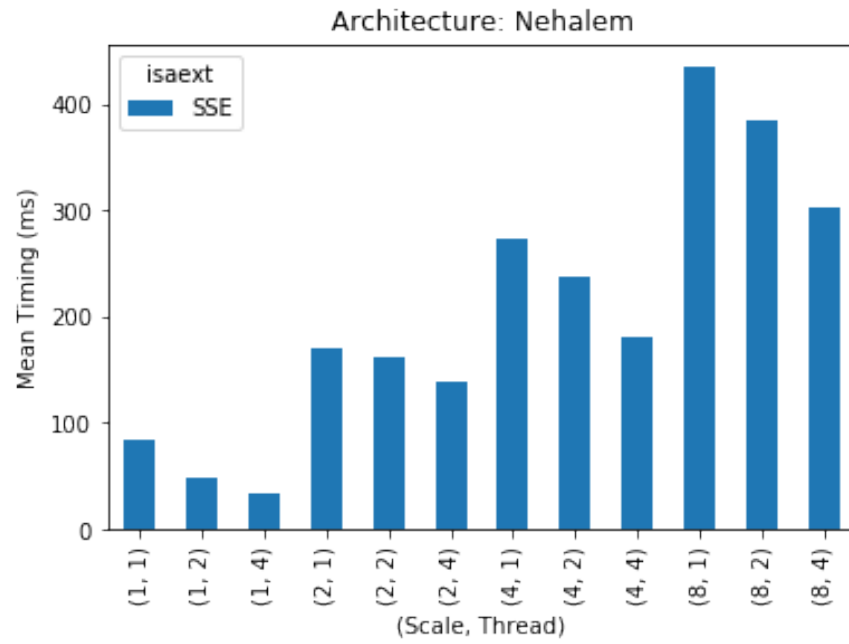


Figure 3

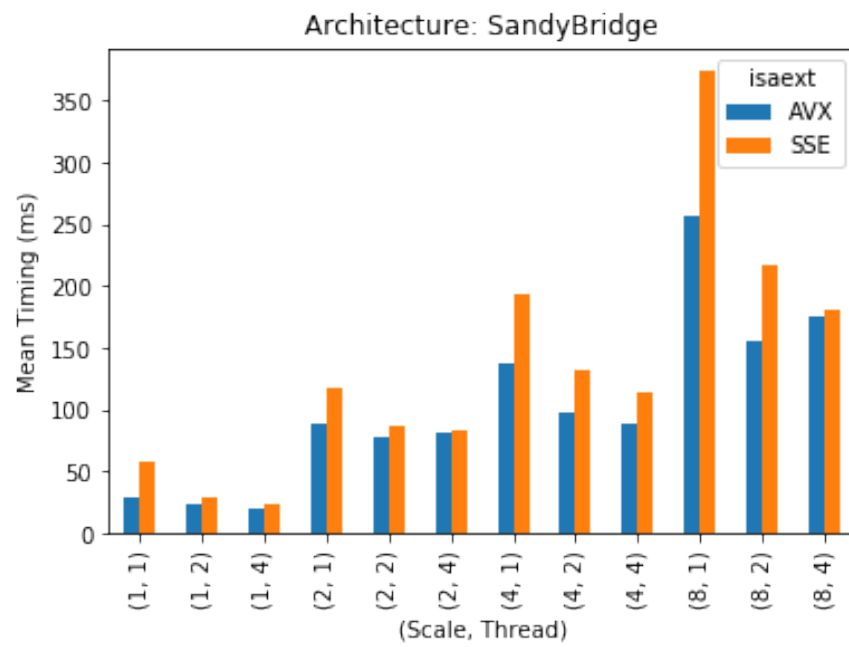


Figure 4

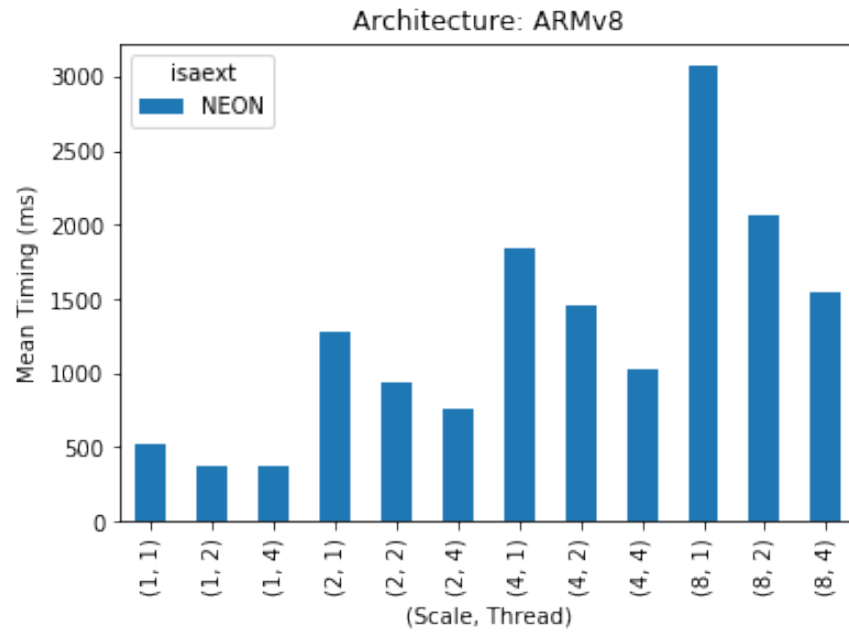


Figure 5

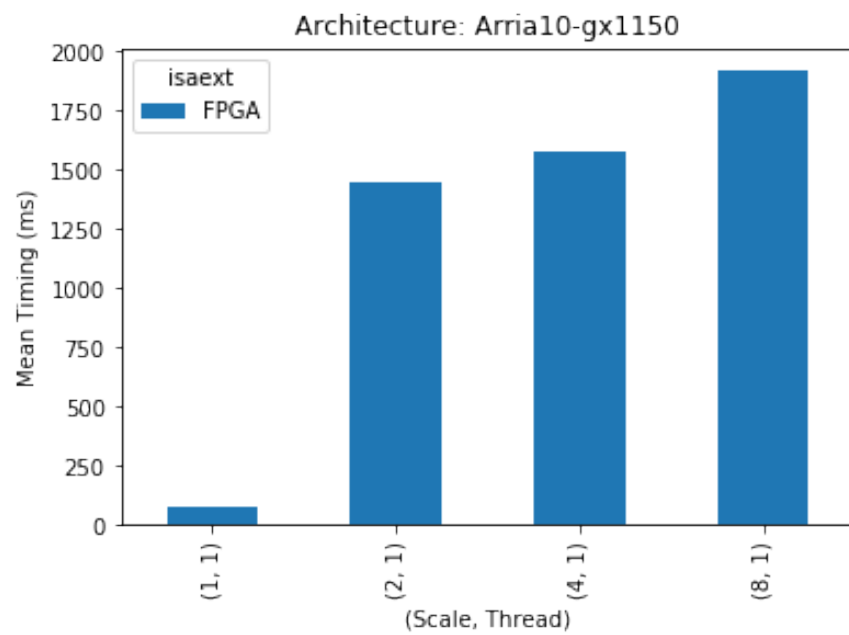


Figure 6

The performance of the system increases with thread number and extra instruction sets across the board. Organizing and maintaining connection with the external jail to the Lincoln Lab AWS clusters has proved to be inconvenient. This is combination with the deadlocks in the interaction model upon failure made the challenge problem more difficult to test than the other challenge problems.

2.3 PRESERVATION OF PERFORMANCE

The third challenge problem extends the capabilities of SPIRAL to detect and suggest appropriate changes in order to maintain the performance of the system. This capability is demonstrated by detecting that the performance (e.g. runtime) of the system is not meeting the deadline specified, and suggesting appropriate changes to the user. For example, changes might include a request for more/less threads, switching to a different instruction set or switching to a different up-sample scale.

2.3.1 Testing Strategy

For this challenge problem, the system is first assumed to be running in single thread mode. The perturbation introduced will be another application running on either the same core or different core. The system should detect that the performance has decreased when the perturbing application is run on the same core, and should prompt the user with possible changes to maintain the performance. When the perturbing application is run on a different core, the system should not prompt the user for a change to the application. The perturbation should be removed after a certain period and the system should detect that it has excess resources. The system should prompt the user to release the excess resources. For evaluation without a human-in-the-loop the user is replaced by a script that automatically accepts each recommendation until the system identifies one for which meet the specification.

2.3.2 Results and Analysis

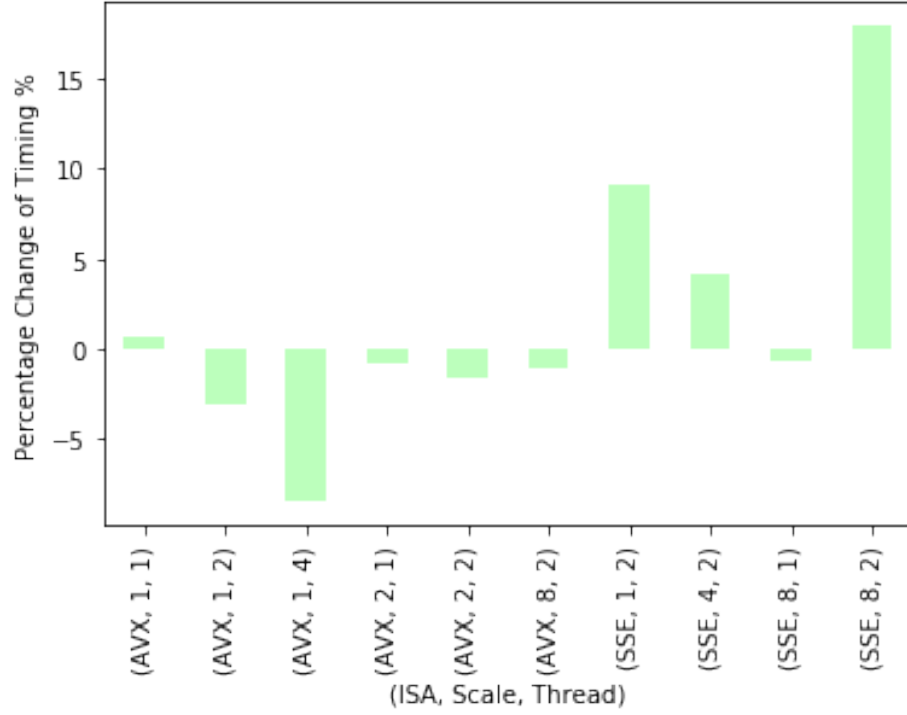


Figure 7. Performance change on increased load

The system appears to achieve preservation of performance. The timing is largely maintained to within 5% across different architectures, loads and configurations.

REFERENCES

- [1] MIT Lincoln Laboratory, “BRASS Phase 2 Evaluation Methodology,” (Sep 20, 2017), URL https://github.mit.edu/brass/mitll/blob/master/documents/evaluation/evaluation_methodology.md.

©2018 Massachusetts Institute of Technology.

Delivered to the U.S. Government with Unlimited Rights, as defined in DFARS Part 252.227-7013 or 7014 (Feb 2014). Notwithstanding any copyright notice, U.S. Government rights in this work are defined by DFARS 252.227-7013 or DFARS 252.227-7014 as detailed above. Use of this work

other than as specifically authorized by the U.S. Government may violate any copyrights that exist in this work.

Massachusetts Institute of Technology
Lincoln Laboratory

**Building Resource Adaptive Software Systems - Phase 2 Evaluation
CRA - Probabilistic Representation of Intent Commitments to Ensure
Software Survival (PRINCESS)**

*Timothy Braje, Robert Cunningham, Jeffrey Hughes, Eric Kilmer,
Timothy Meunier, W. Konrad Vesey, Philip Zucker, Martine Kalke*

September 5, 2018

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

This material is based upon work supported by the Defense Advanced Research Projects Agency under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency.

Lexington

Massachusetts

	CP1	CP2 (Cprog)	CP2 (DSLprog)	CP3
Intent Discovery				
Correct	✓-	✓	✓	✓
Complete	✓	✓	✓	✓
Specific	✓	✓	✓	✓
Fidelity	0.857	0.829	0.409	0.703
Test Coverage	small sample	small sample	complete	medium sample

TABLE 1

A summary of BRASS results for CRA.

1. EXECUTIVE SUMMARY

The challenge problems provided by Charles River Analytics (CRA) developed their PRINCESS technology to focus on the resiliency of an unmanned undersea vehicle (UUV). In challenge problem 1, PRINCESS must adapt to a range of sensor perturbations that affect its ability to localize and determine the UUV’s position to conduct a successful resupply mission. In challenge problem 2, existing mission software must be ported to a processor board (system) of a new instruction set architecture that is inserted into the UUV. In challenge problem 3, the PRINCESS must adapt to battery power perturbations that affect the UUV’s reconnaissance mission. CP1 and CP3 are online problems, so tests were executed to ensure that adaptations performed at least as well as the perturbed (with no adaptation) stage. CP2 was an offline problem, so tests involved verification that the adaptation techniques worked as intended.

1.1 SUMMARY RESULTS AND RECOMMENDATIONS

Each platform and challenge problem is assessed on a number of evaluation criteria which are described in the BRASS Phase 2 Evaluation Methodology [1]. The Phase 2 Evaluation Methodology broadly addresses the general testing strategy as applied to all performers. In Table 1, we produce a summary of some of the metrics called out in that document.

During testing, there were some difficulties in reproducing the results achieved by CRA for CP1 and CP3, and after lengthy communication and debugging sessions, we resolved some but not all of the questions and concerns. The CRA UUV system provided a rich API for the evaluators to explore different and interesting scenarios. Some of the bugs found during testing were related to errors and ambiguity related to the documentation, which were only resolved after communication on GitHub. The CP2 evaluation went well and the team was very responsive in detailing the inner-workings of their system and why some tests failed or were invalid. In all, it was a mostly successful evaluation. Our recommendation for the next Phase of the program would be primarily to resolve the inconsistencies between the results that the CRA team is seeing versus what was delivered and can be reproduced through the docker containers. Furthermore, clearer documentation for

valid parameter constraints and values should be cleared up for all challenge problems for better understanding by new team members and future evaluators.

2. DETAILED RESULTS

2.1 ONLINE SENSOR ADAPTATION AND STATE ESTIMATION

In this challenge problem, an Unmanned Undersea Vehicle is engaged in a resupply mission to travel to a rendezvous point with a resupply vessel. The challenge problem seeks to determine PRINCESS’s capability to adapt to a range of perturbations that affect its ability to localize and determine its position, which is necessary to conduct the resupply mission. These perturbations include degradation of sensors used by the UUV as well as perturbations to the UUV’s environment. Given one or more perturbations, PRINCESS needs to determine the source of the problem and adapt the appropriate component. This may involve synthesizing a new data adapter for the sensor in real time and/or reconfiguring the settings of the Kalman filter or other algorithm used for localization.

2.1.1 Testing Strategy

Test design for this challenge problem involves four different scenarios, with the first three written by the CRA team [2]:

1. Stuck-at: Consecutive sensor readings that have zero variance and are different from normal sensor readings. The perturbation lasts until the end of the scenario.
2. High-noise: Consecutive sensor readings that have an unexpectedly high variance. The perturbation lasts until the end of the scenario.
3. Mis-calibration: A time window of sensor readings with a constant offset from the ground-truth values. The perturbation lasts until the end of the scenario.
4. Chaos: This is Lincoln’s own test that combines all of the above 3 tests into one execution.

Since the parameter space is so large, we try to execute a number of tests for each of the scenarios that we feel shows a meaningful set of PRINCESS’s results.

2.1.2 Results and Analysis

The verdict expression is calculated based on how close the UUV comes to the destination. Specifically, if the UUV is less than 75 meters from the destination, the verdict is Pass, else Fail. Thus, a useful way to visualize the verdict is to graph where, for each complete test case, we plot the distance from the destination on the x-axis against the test group identifier along the y-axis. We split the graphs into sub-graphs based on the test case, as described above. After running 40 tests (10 for each test case), we can see in Figure 1 that most tests from the Adapted stage perform better than the Perturbed stage and are within the 75 meters (vertical blue line) for test cases 1, 3, and 4. However, for test case 2, after conversing with the CRA team, the results are much different

than expected, and we were unable to debug further due to time constraints. Table 2 shows the results obtained by CRA for the test case 2 as compared to what we found.

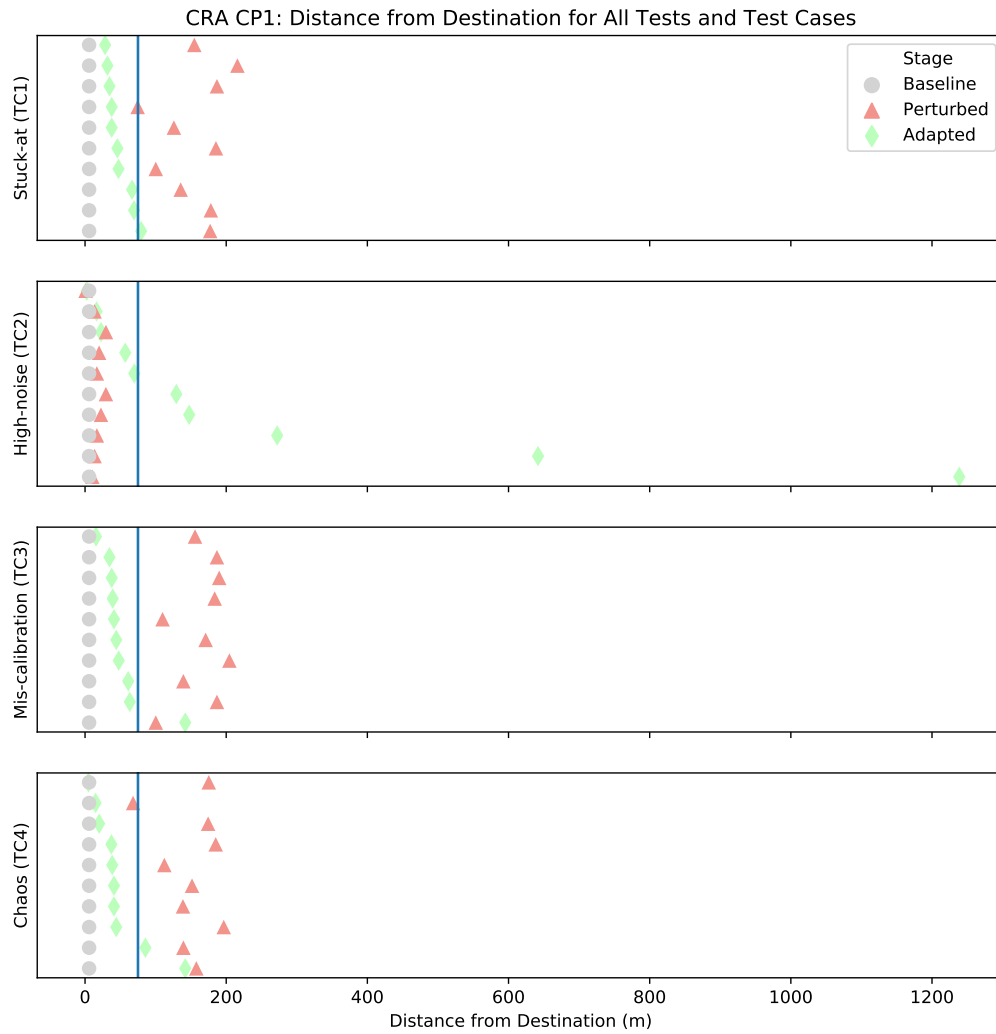


Figure 1. UUV distance for all tests separated by each test case for CP1. The vertical blue line indicates 75 meters, as required for a Pass verdict.

TABLE 2
Contented Results CP1: Test Case 2

Test	Baseline	Perturbed	Adapted	Adapted (LL)
0	5.76	0.75	0.85	2.55
1	5.76	13.52	16.74	16.37
2	5.76	29.58	29.58	22.69
3	5.76	19.94	16.74	57.01
4	5.76	16.76	16.77	69.84
5	5.76	29.57	29.57	129.27
6	5.76	22.56	41.32	147.57
7	5.76	16.76	16.77	272.22
8	5.76	13.53	13.54	641.81
9	5.76	10.31	10.33	1238.63

It is worth noting that a few days of debugging were required to obtain the results in Figure 1. Initially, we believe that the SUT was being starved of compute resources, which led to the Perturbed and Adapted distances being very close to equal—not ideal. Eventually we upped the computing resources to 8 CPU cores, and 24GB of RAM, which produced the results shown. As the program continues, it will be important for the CRA team to figure out why resource starvation lead to different results instead of either an outright crash or increased test duration (with the same results shown in this document).

Looking at the outcome and verdict plots in Figure 2, we see that almost all of the tests complete and pass. Note that this graph does not include the contentious results from test case 2. Of the 2 tests that were Invalid, one test was from each of test cases 1 and 4, but other than that the cause is not obvious and would need to be looked at closer. Similarly, the 4 tests that were Complete with verdict Fail came from all of the test cases, and considering how the Perturbed results for each of the tests were not the worst performers of their respective test cases, the cause is not obvious and would require further analysis. Furthermore, it is not apparent why the Chaos test case (that includes all of the other test cases) did not track or seem to be affected by the behavior seen in test case 2.

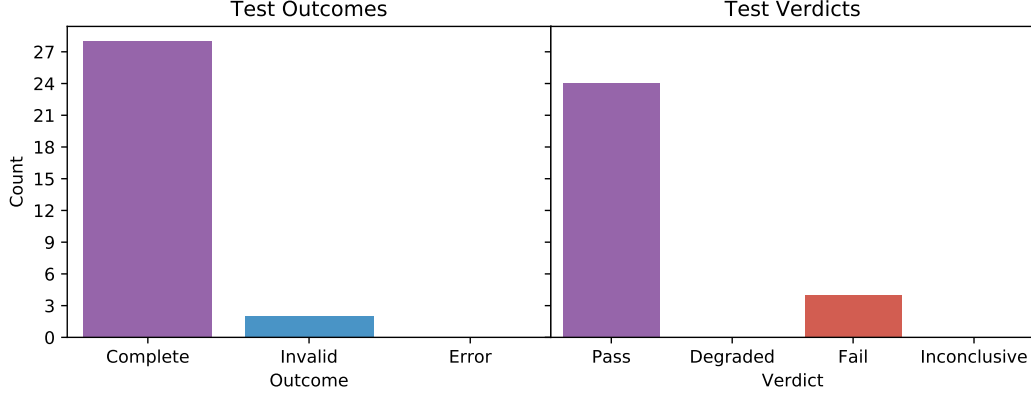


Figure 2. Verdicts and outcomes for CP1, not counting test case 2.

2.2 NEW COMPUTING PLATFORM

In this challenge problem, the processor board (system) on which the UUV mission payload software runs is changed. This requires a port of the mission software to an operating system that runs on the new system. Where tools exist to target the new instruction set architecture (ISA), this could be as simple as setting some compiler flags or as difficult as rebuilding the entire system, from the ground up – including producing a new compiler toolchain (i.e., assembler, linker, loader), operating system, and mission software. Where the processor has no or poor existing tool support, an adaptation may be impossible until adequate tools and system libraries are available. PRINCESS facilitates adaptation by synthesizing the tool chain and operating system from descriptions of the new processor and processor board to enable the mission payload software to run on the new UUV hardware platform.

This challenge problem runs offline – while the UUV is docked. There is no automated detection of perturbations; a change of hardware platform is a planned-for event, and the synthesis of the new software takes place outside the UUV. The newly synthesized software would be then downloaded to the UUV.

2.2.1 Testing Strategy

There are two distinct operating modes for CRA CP2: toolchain (Cprog) and OS synthesis (DSLprog). We were unable to completely test all of the Cprog parameter space, as described below, but we *were* able to test all parameter combinations for DSLprog. These two operating modes do not have any overlap in the values for the fields, so we discuss each of the parameter spaces and additional testing details separately.

2.2.1.1 Cprog

The dataset for Cprog came from the Csmith [3] software tool with a few filters to remove programs that loop forever and others, but the number of different input programs is effectively

infinite. Therefore, given a set of pre-generated C programs, we used a uniform random number to choose which Csmith program to provide as the input. Furthermore, there were two different ISA to choose—32-bit ARM EABI and 64-bit SPARC—and we chose the ISA randomly to pair with the random C program.

2.2.1.2 DSLprog

The dataset for DSLprog consisted of 6 hand-written specifications in the Alewife domain specific language. These handwritten specifications defined the abstract behavior of 6 real OS module snippets from BarrelfishOS, a legacy open source OS kernel. Note that randomly generated specifications would not represent actual OS module specifications and hence would be out of scope for CP2. CRA provided two sets of 4 perturbed machine descriptions written in their Cassiopeia DSL. Each machine description modeled either the x86_64 or ARMv7-A architectures with between 1 and 4 CPU registers. We tested all of these cases except test snippet numbers 0 and 1 that always failed (sanity checks) and other test snippet/perturbation combinations¹ that were not applicable.

2.2.2 Results and Analysis

Figure 3 shows the outcome and verdict graph of the Cprog system. We can see that there are a few failures. After presenting the results to the CRA team, we learned that in some cases these failures are related to “SPARC64 instructions with a discontinuous field, which the linker needs to be able to patch at link time from relocations. Discontinuous fields are themselves OK, but the linker doesn’t know how to handle them, so this doesn’t work, and values that don’t fit in the lower portion of the field cause the linker to error out.”² One benefit of this testing was that the CRA team learned that a forward symbol reference of a certain form, on the ARM architecture, was illegal, but the results showed it was in fact legal! Of the Invalid test cases, these were likely due to unsupported properties in the C programs used as input that were not caught by the filters, however the exact property is not obvious and requires more analysis.

¹ <https://github.mit.edu/brass/cra-princess/issues/81#issuecomment-33594>

² <https://github.mit.edu/brass/cra-princess/issues/81#issuecomment-33605>

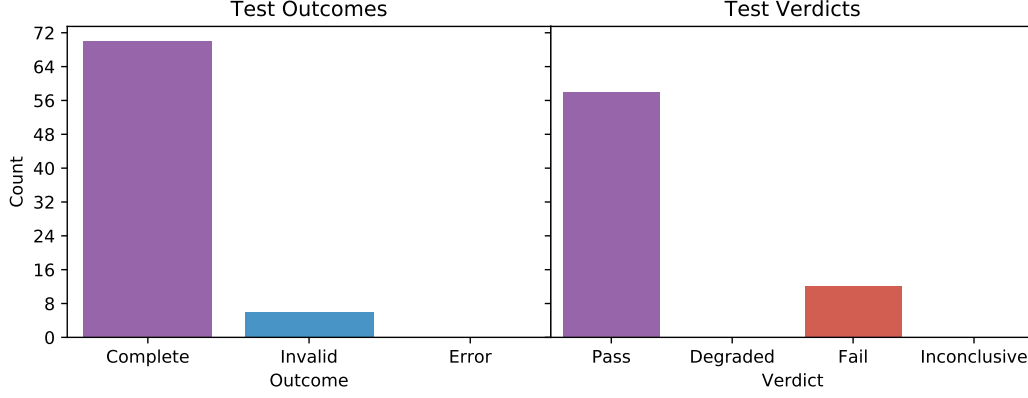


Figure 3. Verdicts and outcomes for CP2 Cprog.

Figure 4 shows the outcome and verdict graph of the DSLprog system. We can see that there are more failures than passes. After discussing with the CRA team, we were told that all of these failures are due to a few reasons: resource starvation, an incorrect BarrelfishOS implementation (cannot prove that is correct), and an optimization/timeout change (now fixed in future implementations!).³ In the absence of these confounding factors, all other tests were successful.

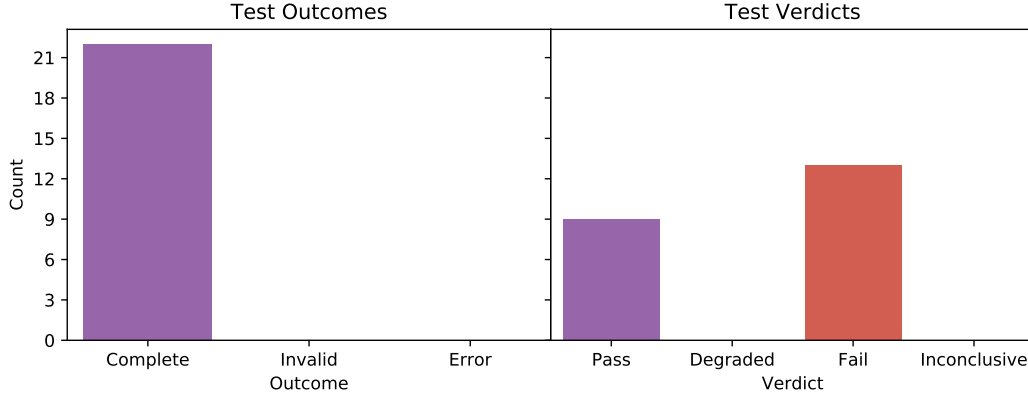


Figure 4. Verdicts and outcomes for CP2 DSLprog.

2.3 MISSION-LEVEL ADAPTATION AND VERIFICATION TO POWER FAILURES

This challenge problem involves an unmanned undersea vehicle (UUV) conducting a reconnaissance mission in a limited energy scenario. While conducting the mission, it is important for the UUV to conserve enough energy that it can return to its home base safely, otherwise it is at

³ <https://github.mit.edu/brass/cra-princess/issues/81#issuecomment-33594>

risk of being abandoned at sea. The UUV is buoyant, so it will not sink, but if it cannot return home it becomes abandoned and may be recovered by the adversary, which would be a significant loss. When the UUV was tasked with its mission, there was sufficient energy for it to complete the mission safely and return home. The perturbation in this challenge problem is the loss of some of its energy capacity. Given this perturbation, the UUV needs to decide whether to continue to conduct the mission as planned, to alter mission parameters, to return home immediately, or to reconfigure some of the parameters of the software. These considerations describe a complex optimization space.

The proposed challenge problem differs from our Phase 1 challenge problems in two main respects. First, in Phase 1 we adapted only a single software component (the Kalman filter), whereas in Phase 2, we will adapt the whole navigation system and the mission. Adaptation can happen at multiple levels of the software hierarchy. At a component level, the UUV can decide to shorten the maximum search path length, potentially saving energy but reducing the likelihood of detecting the object. At the mission level, the UUV can adjust the thresholds for minimum energy remaining and minimum acceptable probability of object detection as a result of perturbations that cause the original mission to be infeasible.

2.3.1 Testing Strategy

For this challenge problem, we must ensure that the object is placed within the search area. The search area has a range of longitude and latitude for where it is placed, so we first choose a random search location box within these bounds, and then we randomly place the object in the upper half of the box. The object is placed in the upper half of the search region to only test scenarios that would be interesting and difficult, since the UUV starts in the lower left corner of the search region. As for choosing battery perturbations, we first choose a random number of perturbations, then, for each perturbation, we choose a random number between a predefined range of battery drop percentages. The times at which the battery perturbations occur is randomly chosen between a predefined (by CRA) time range. Since the parameter space is so large, we randomly sample throughout the entire parameter space. Exact parameter values are described in [4].

2.3.2 Results and Analysis

The verdict expression for CP3 is calculated based on two intents: 1. Does the UUV return with power? 2. Does the UUV find the object? A Pass is given if both intents are met, a Degraded is given if the UUV only returns with power, and a Fail is given if the UUV does not return (due to loss of power) regardless of if the object was found. The information returned by the SUT in the `/done` message consists of two fields describing the result of the two intents and also the location of the found object.

We try to visualize the results of our tests using two scatter plots based on the test stage (Perturbed and Adapted) and the object's location on a coordinate field, as shown in Figures 5 and 6.

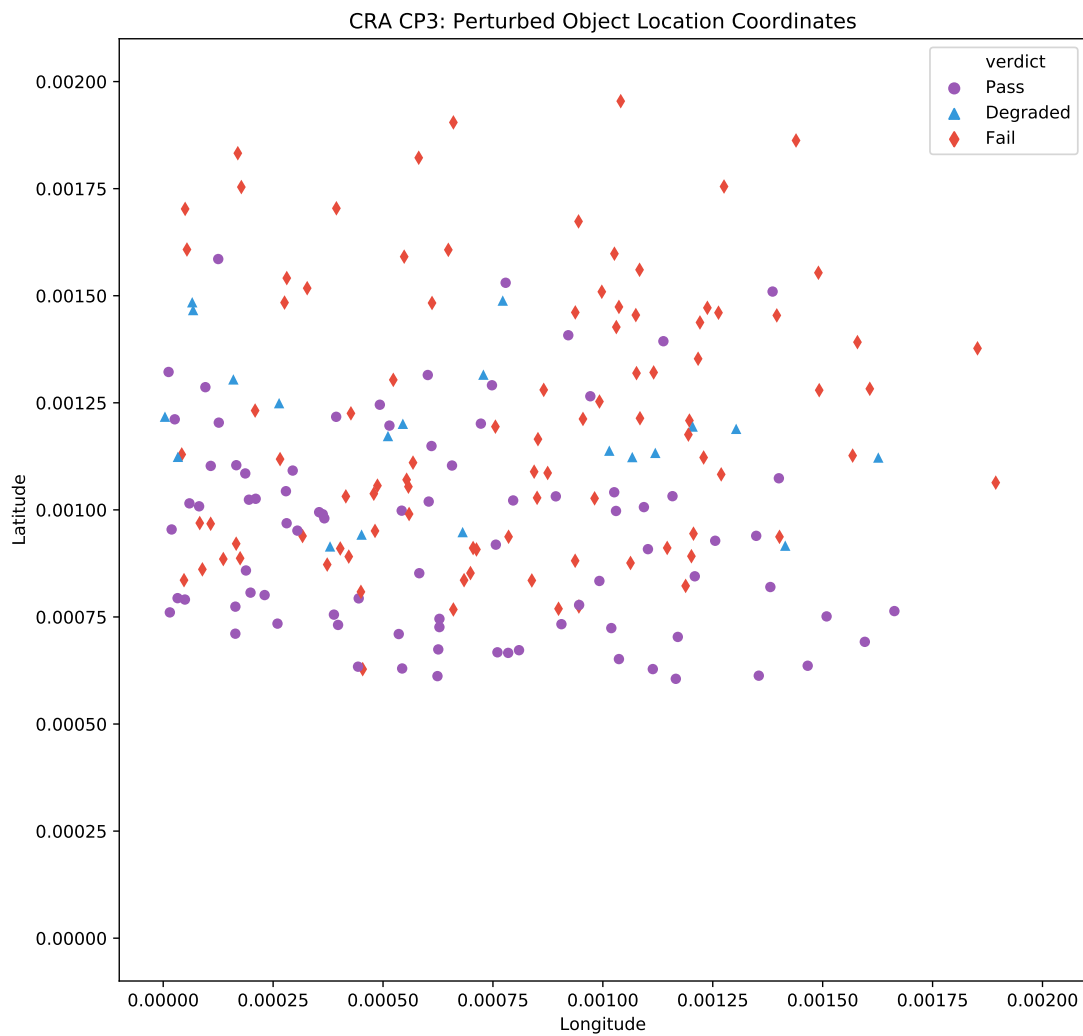


Figure 5. Object location and verdict for all tests in the Perturbed test stage.

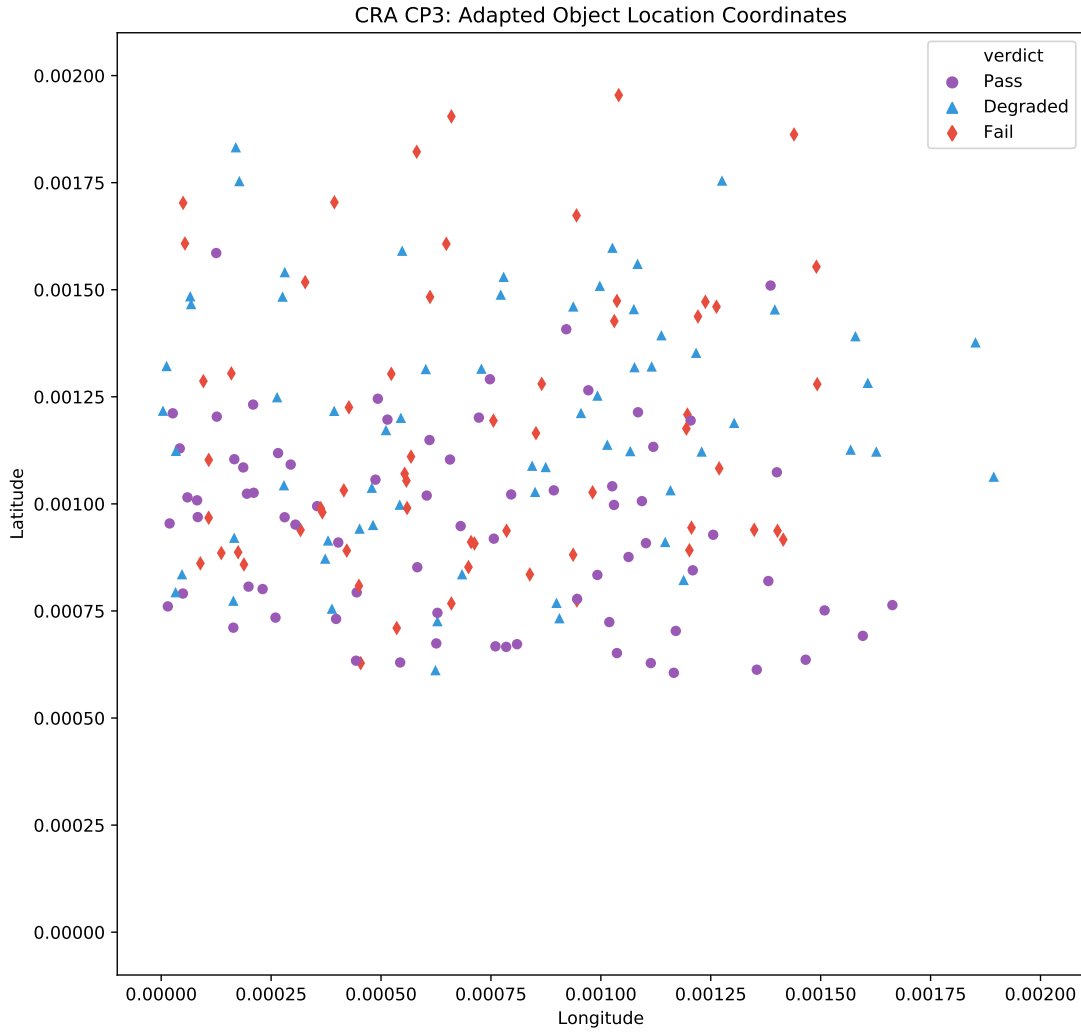


Figure 6. Object location and verdict for all tests in the Adapted test stage.

Note that the object location and verdict result plots do not indicate the number or severity of the battery reduction perturbations.

The bar charts in Figure 7 also help to visualize the difference between each test stage.

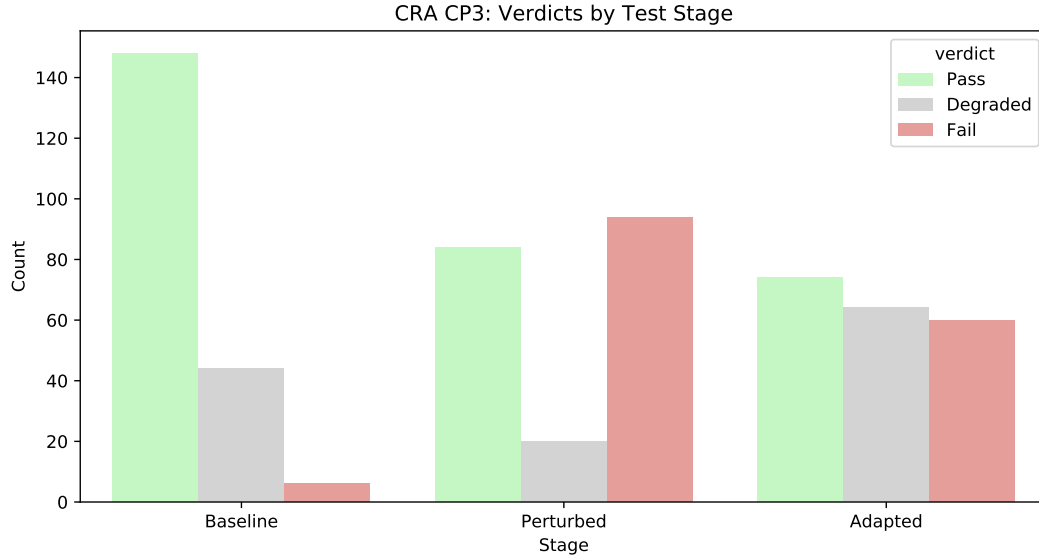


Figure 7. Bar graph of each test verdict with respect to the test stage.

As we can see, the difference between Perturbed and Adapted test stage leads to a decrease in Fail verdicts, an increase in Degraded verdicts, but, oddly, a decrease in Pass verdicts. We would have expected the number of Pass verdicts in the Adapted stage to be equal or greater than the number of Pass verdicts in the Perturbed stage. One explanation that the CRA team gave for this is that the Perturbed stage could have found the object earlier than the Adapted stage (due to pathing differences from adaptation) and returned home before the second (or more) perturbation hit. However, the main take-away here is that the number of Fail verdicts decreased and the number of Degraded verdicts went up, meaning that the UUV was able to make it back without losing all power.

Figure 8 shows the standardized outcome and verdict plot, and it is great to see that almost all tests are Complete. The only Invalid tests come from when the Baseline stage was a Fail. The cause of the Baseline stage failing is unknown at this point.

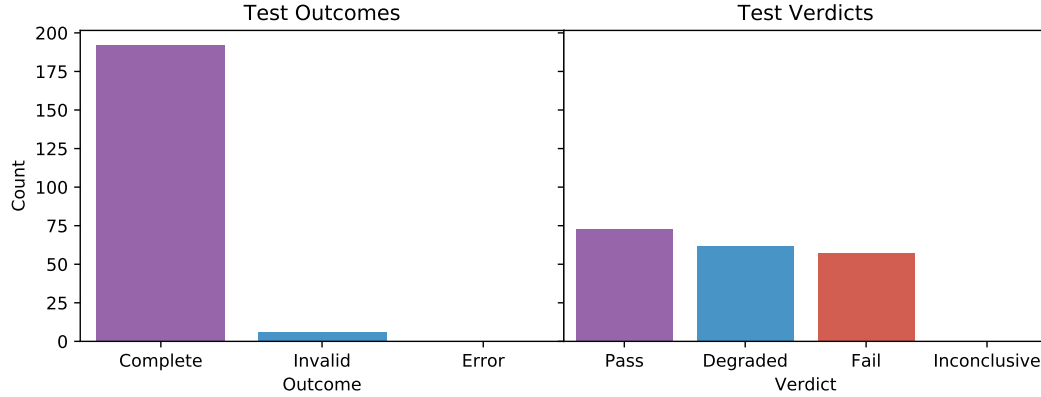


Figure 8. Verdicts and outcomes for CP3.

REFERENCES

- [1] MIT Lincoln Laboratory, “BRASS Phase 2 Evaluation Methodology,” (Sep 20, 2017), URL https://github.mit.edu/brass/mitll/blob/master/documents/evaluation/evaluation_methodology.md.
- [2] “Charles River Analytics PRINCESS: Online Sensor Adaptation and State Estimation,” https://github.mit.edu/brass/cra-princess/blob/master/Phase2/CP%20Descriptions/CP1/CRA%20CP1%20Challenge_Problem_Announcement_v17.docx, accessed: 08/23/2018.
- [3] X. Yang, Y. Chen, E. Eide, and J. Regehr, “Finding and understanding bugs in c compilers,” *SIGPLAN Not.* 46(6), 283–294 (2011), URL <http://doi.acm.org/10.1145/1993316.1993532>.
- [4] “Charles River Analytics Challenge Problem 3: Mission-Level Adaptation and Verification to Power Failures,” https://github.mit.edu/brass/cra-princess/blob/master/Phase2/CP%20Descriptions/CP3/CRA%20CP3%20Challenge_Problem_Announcement_v18.docx, accessed: 08/23/2018.

©2018 Massachusetts Institute of Technology.

Delivered to the U.S. Government with Unlimited Rights, as defined in DFARS Part 252.227-7013 or 7014 (Feb 2014). Notwithstanding any copyright notice, U.S. Government rights in this work are defined by DFARS 252.227-7013 or DFARS 252.227-7014 as detailed above. Use of this work other than as specifically authorized by the U.S. Government may violate any copyrights that exist in this work.

Massachusetts Institute of Technology
Lincoln Laboratory

**Building Resource Adaptive Software Systems - Phase 2 Evaluation
Rice - Proteus: Controlling Resource-Adaptive Embedded Software**

*Timothy Braje, Robert Cunningham, Jeffrey Hughes, Eric Kilmer,
Timothy Meunier, W. Konrad Vesey, Philip Zucker, Martine Kalke*

September 5, 2018

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

This material is based upon work supported by the Defense Advanced Research Projects Agency under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency.

Lexington

Massachusetts

1. EXECUTIVE SUMMARY

The BRASS program supports research to support the development of long-lived systems. Such systems must endure different types of dynamic perturbations, including mission change, the deterioration of hardware, and other situations where the system is used under operational conditions that differ from those the system was designed and tested for. Continuing to operate correctly under such change requires the ability to adapt dynamically as well as means to specify *intent*, that is, what “operate correctly” means.

In Phase 1 the Proteus system, a framework which enables dynamic behavior in embedded applications, was developed. Proteus uses a combination of control theory and machine learning to select system knobs (parameters) in order to maintain a desired level (meet a constraint goal) of application behavior such as performance, or adapt to a requested change in performance, in response to changes in available processor capabilities (i.e. number of processor cores or maximum operating frequency). This is done while optimizing an objective function, such as quality or power consumption, which rules out using a fixed over-provisioned configuration to guarantee that the desired intent is met. Proteus uses machine learning to model the relationship between system configuration and system behavior, which can be nonlinear, and complements these models with feedback control to adapt to dynamic environmental changes, such as ambient system load. The combination of machine-learned models and control theory also lends itself to reasoning about the possible behavior of the system, such as what behaviors the system can achieve.

Dynamic adaptation requires identifying configurations that suit the current application, mission and environment. Doing so too often (such as, for every frame of a video) incurs too much overhead to be feasible for real-time applications, which are common in the embedded domain. The solution is to compute a schedule of configurations to be applied over a *window* of time into the future. Figure 1 shows that doing so by brute force, considering all possible schedules, is intractable even when the system has a very small number of possible configurations, and for very small windows. Proteus deals with this challenge by exploiting knowledge about the structure of intents that allows it to consider a small enough subset of the schedule space during execution, for the scheduling to complete sufficiently fast to make real-time operation possible. The dramatic reduction in size from the total number of schedules to the number of feasible schedules (that approximately meet the constraint) reflects the difficulty of just solving the constraint part of an intent. The further reduction in size, going to the number of near-optimal schedules shows that it is even more challenging to solve the constraint while maintaining close to optimal behavior.

Phase 1 focused on tuning the performance of the application at runtime to achieve a set performance goal. It achieved this by exploring the space of available system knobs. Phase 2 builds on this in two notable ways 1) it allows richer expression of intent (related to higher-level mission goals) and 2) the possible adaption space is extended to include application specific knobs. Richer expression of mission intent is now possible based on the team’s work to develop the FAST programming language which includes application intent specifications in addition to conventional code. Using these specifications, it is possible to constrain any measure (such as performance or power consumption) exposed by the challenge problem, and use an expression formed from these measures as the objective function to be optimized during execution (such as to minimize power

consumption or maximize the quality of output). These specifications can also be changed dynamically, without having to change the models that Proteus uses to control the challenge problems.

In Phase 2, three streaming signal processing applications were selected to demonstrate the capabilities of the system. The first two, synthetic aperture radar and x264 video encoding, were explored in Phase 1 and have been extended for use in this Phase. The third challenge problem, CaPSuLe, represents an image matching problem which is new for Phase 2. To facilitate testing in software, the evaluation relies on a database driven emulator which contains samples measured from an embedded platform’s performance characteristics and energy consumption, as opposed to an evaluation conducted directly on embedded hardware. The emulator has been updated in Phase 2 to account for a new embedded hardware platform, a Xilinx ZCU architecture, additional application knobs, and challenge problems.

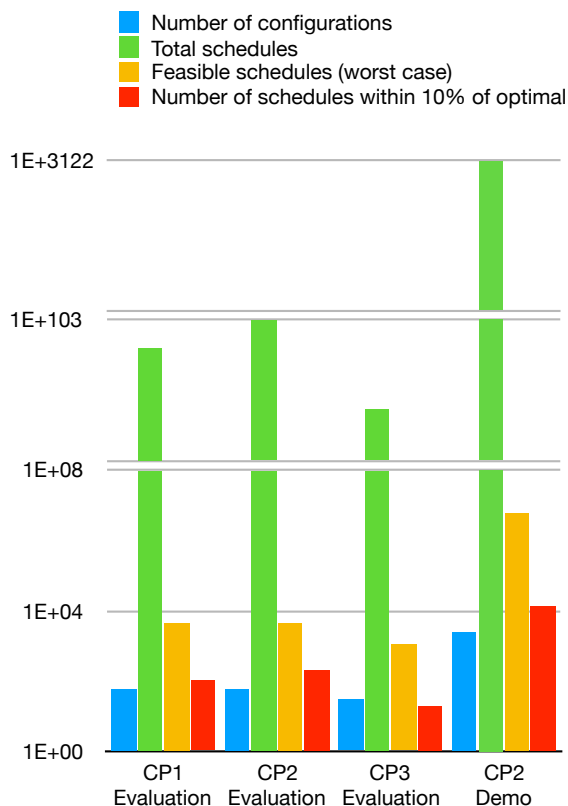


Figure 1. This chart illustrates the complexity of the scheduling problem that Proteus solves (as illustrated in Figure 2). The four bar groups represent the numbers of configurations, brute-force schedules, feasible schedules (considered by Proteus), and near-optimal feasible schedules. The relatively small number of configurations included in each of the evaluation models ensures that the computation of the corresponding oracles remains tractable.

	CP1	CP2	CP3
Intent Discovery			
Correct	✓	✓	✓
Complete	✓	✓	✓
Specific	✓	✓	✓
Fidelity	0.826	0.979	1.0
Test Coverage	medium sample	medium sample	medium sample

TABLE 1

A summary of BRASS results for Rice.

1.1 SUMMARY RESULTS AND RECOMMENDATIONS

Each platform and challenge problem is assessed on a number of evaluation criteria which are described in the BRASS Phase 2 Evaluation Methodology [1]. The Phase 2 Evaluation Methodology broadly addresses the general testing strategy as applied to all performers. In Table 1, we produce a summary of some of the metrics called out in that document.

The test plan developed together with Rice [2] for the Phase 2 evaluation was designed to validate that Proteus is able to meet the intent (both in terms of constraint and in terms of objective function) provided to it initially, as well as through dynamic perturbations. These perturbations include changes to the intent itself, and in the additional test parameters included for Phase 2. These parameters represent dynamic failures in the hardware (number of cores and the core frequencies that available to perform computations), changes to the mission length, and changes to the degree to which the output quality should be prioritized (“scene importance”).

Overall, the tests went very well. The team from Rice was very helpful in their communication and debugging efforts while also providing insight into the results produced by Proteus. The API provided us sufficient control of the system to explore and test many different and useful scenarios. During testing, we discovered some results that we were not expecting, but we were quickly able to remedy the cause and proceed with additional testing. The test plan that the Rice team helped write was useful in executing interesting test cases. In all, it was a successful evaluation! Our recommendation for the next phase of the program would be to fully implement, with the ability to execute, all three test stages—Baseline, Perturbed, and Adapted—instead of just the Perturbed and Adapted stages to get a more explicit indicator of the capabilities of Proteus.

2. DETAILED RESULTS

2.1 SYNTHETIC APERTURE RADAR

The first challenge problem represents a streaming signal processing application which was also explored in Phase 1. The context of the challenge problems is an embedded system, e.g. an

unmanned aerial vehicle (UAV) equipped with various sensors that is conducting a designated mission. The UAV must process the incoming data from the sensors and report its analysis back to base. The available resources may fluctuate, for example, battery level or processor availability. The system attempts to adapt to these changes by utilizing online control and machine learning supported by off-line machine learning in order to maintain the quality requirements posed by the base by leveraging the capabilities of FAST, a new programming language that includes intent specification in addition to conventional code. The aforementioned tasks are carried out by a dedicated on-board system (independent from the avionics control) running the corresponding intent-enriched application.

Compared to Phase 1, two additional aspects of this scenario are reflected in the challenge problem. The effect of mission length on achievable accuracy is captured by the additional test parameter `missionLength` and the effect of priority (the “importance” of the scene currently processed by the application) on the quality of the output is captured by the additional test parameter `sceneImportance`. The goal of this challenge problem is to demonstrate that Proteus is able to meet intents by adapting combinations of system configuration knobs and application knobs, which was not demonstrated in Phase 1.

2.1.1 Testing Strategy

For this challenge problem, we followed a test document that was written in collaboration with the team from Rice. This test document described eleven different scenarios where each scenario targeted different parts of the system and represented real-world or common cases. While each of the test cases is designed to be challenge-problem independent, the parameter values are specific. While the official evaluation plan [2] describes the test cases in more detail, we list a short title for each test case here:

1. Jump in constraint goal
2. Change between constraint measures
3. Invert the optimization type and negate the objective function
4. Negating the objective function
5. Multiple jumps in constraint goal
6. Two consecutive jumps in constraint goal without enough time to meet the intent (should fail)
7. Constraint goal not achievable (should fail)
8. Increase in mission length
9. Increase from low to high scene importance
10. Disable half cores
11. Limit the core frequency

Furthermore, each test case is labeled with either an “I” for an “interesting” test case that showcases/validates normal system behavior, or a “C” which stands for “corner case”, where the system executes outside of intended operating conditions. The two corner cases we tested are cases 6 and 7.

These test cases validate several new features compared to the system that was evaluated in Phase I:

- Test case 1-7 perturb the mission intent parameter, which in Phase II is a program fragment expressed in the FAST Intent Specification Language.
- Test case 2 enacts a change between different constraint measures, testing the capability to change what the system is constraining during the course of a mission.
- Test cases 3 and 4 enact a change in the objective function, testing the capability to optimize different aspects of system behavior during the course of a mission.
- Test case 5, illustrated in Figure 2, enacts multiple changes in the constraint goal, testing Proteus’ general capability for dynamic adaptation, which is not limited to any particular number of perturbations.
- Test case 8, illustrated in Figure 3, perturbs the “mission length” test parameter, which is new for Phase II. Increasing this parameter causes a drop in the **energyDelta** (energy spent to process a single stream element, such as a radar pulse) measure, making it possible to process the longer stream corresponding to the increased mission length.
- Test case 9, illustrated in Figure 4, perturbs the “scene importance” test parameter, which is new for Phase II. Increasing this parameter causes an increase in the (application-specific) **quality** measure.

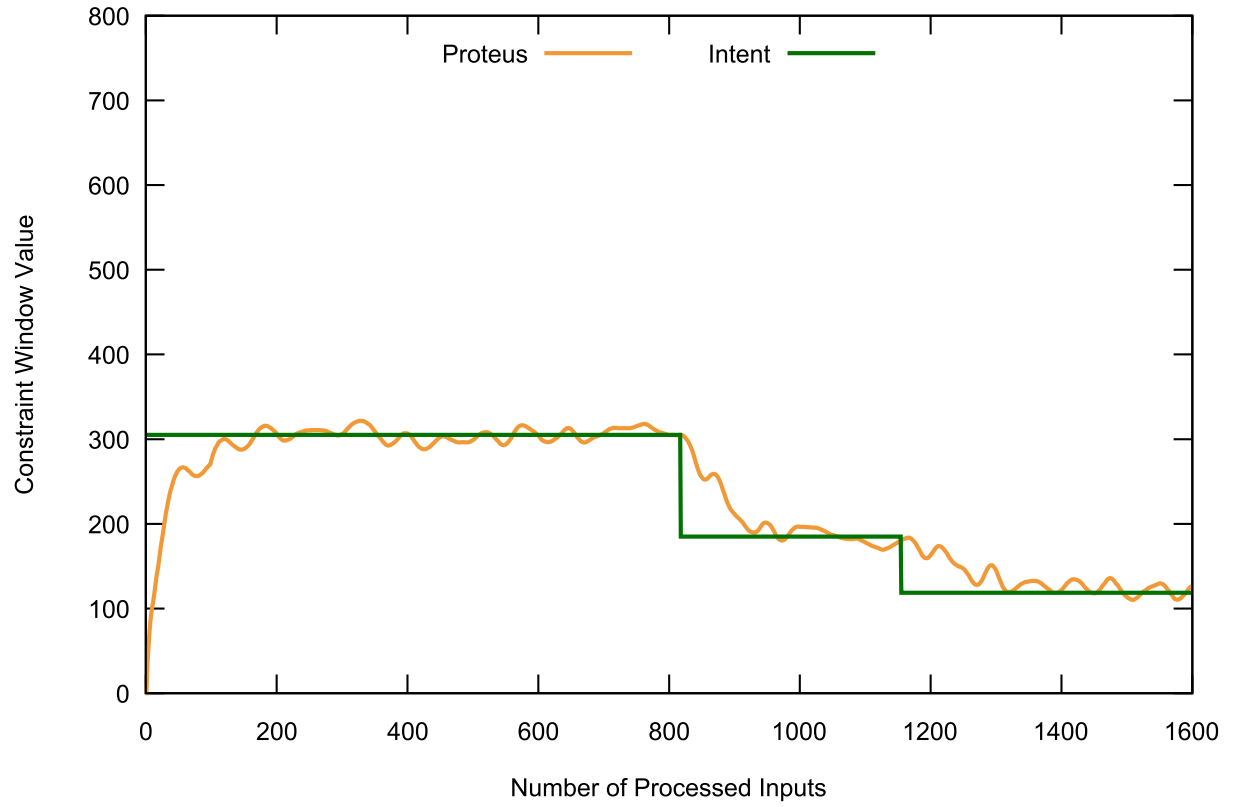


Figure 2. Plot for CP1 showing Proteus (orange line) adapting to multiple changes in the constraint goal (green line). Perturbations occur at input number 818 and 1155.

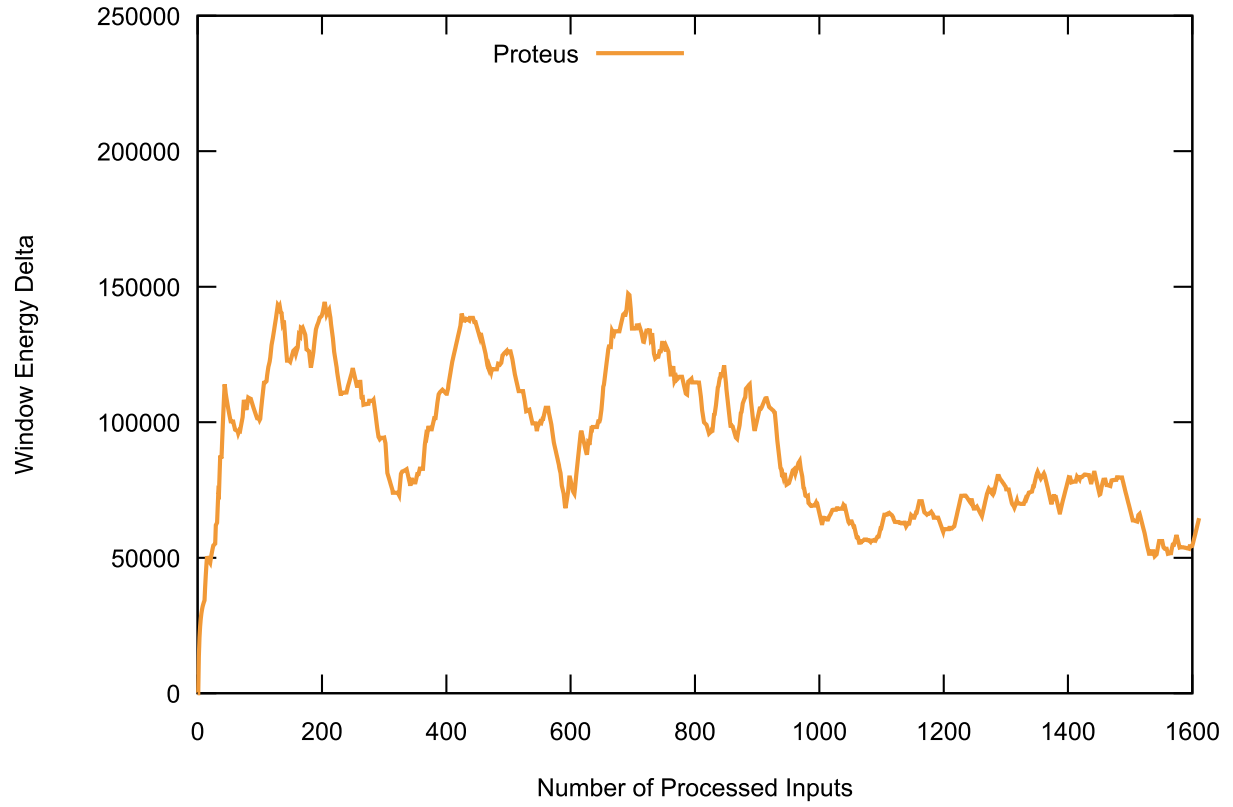


Figure 3. Plot for CP1 showing Proteus (orange line) adapting to an increase in the mission length parameter, resulting in a drop in the energy delta (energy used to process a single radar pulse). The perturbation occurs at input 800.

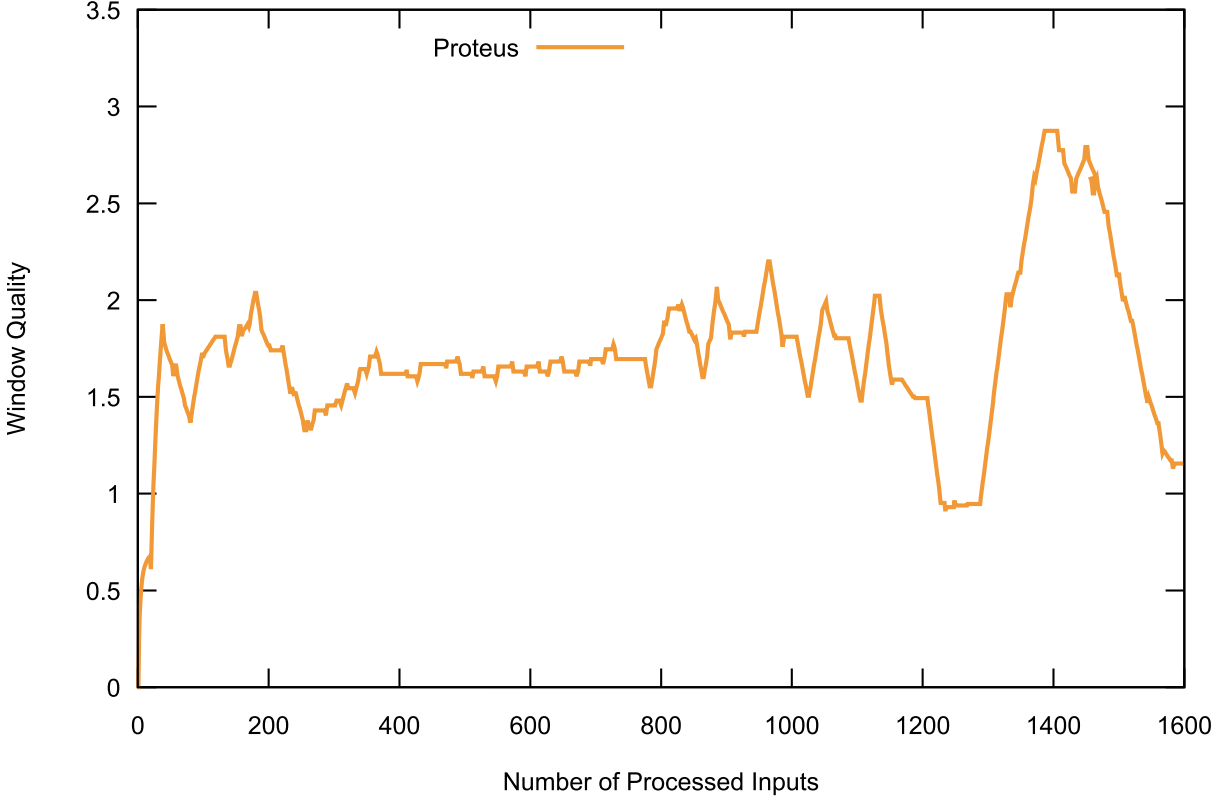


Figure 4. Plot for CP1 showing Proteus (orange line) adapting to an increase in the scene importance parameter, resulting in an increase quality, whenever this is possible while meeting the active intent’s constraint. The perturbation occurs at input 772.

2.1.2 Results and Analysis

The verdict calculation was based on the comparison between the results from an “oracle”, which approximates an ideal adaptive system, and results from Proteus. Figure 5 illustrates this oracle, which is based on a set of traces, obtained by executing the challenge problem in each of the fixed configurations that are available to Proteus.

The oracle’s trace is constructed in an input-wise fashion. At each input, its measures are defined as the measures of the fixed-configuration execution chosen as follows. Consider the relative window errors (versus the constraint goal) of the measures of each fixed configuration. If none of them are below a fixed threshold called `LOCAL_ERROR_THRESHOLD`, we choose the configuration with the lowest such error. Otherwise, we choose the one with error below `LOCAL_ERROR_THRESHOLD`, while having an optimal objective function.

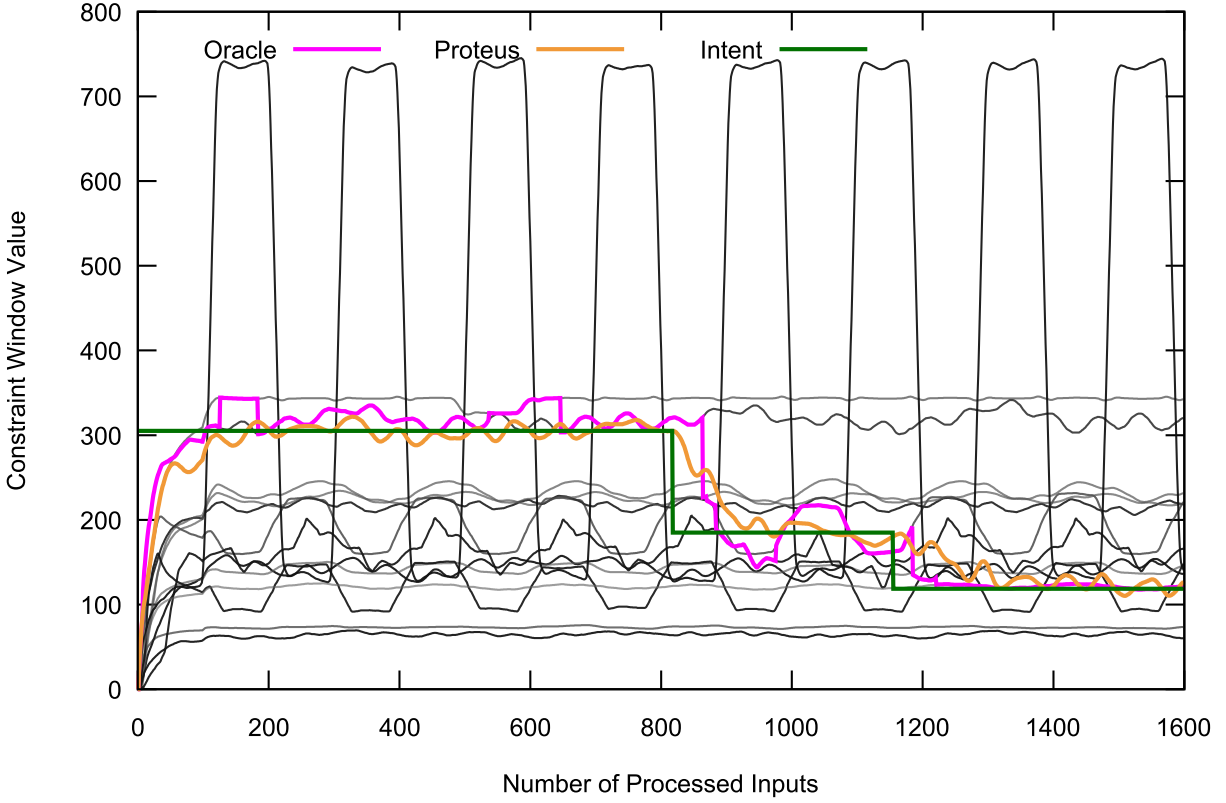


Figure 5. Plot showing Proteus (orange line) and the Phase II oracle (pink line) adapting to changes to the intent’s constraint goal (green line) enacted in test case 5. The grayscale lines correspond to the traces of all the fixed-configuration executions that the oracle is constructed from.

Once the oracle trace has been constructed, it can be used to define a verdict expression, which produces either a **PASS** or a **FAIL** verdict by comparing the traces of the oracle and Proteus executions using two types of statistics, corresponding to two aspects of the system behavior that are taken into account by the verdict expression.

First, the verdict expression checks that Proteus is close enough to meeting the constraint part of the intent. To assess the degree to which an execution (either oracle or Proteus) does this, the SUT returns a “relative cumulative absolute error”. This quantity aggregates the input-wise absolute error of a given execution (either oracle or Proteus), compared to the constraint goal. A small error means that the execution was close to the goal on average.

Second, the verdict expression checks that Proteus optimizes the objective function specified in the intent (while meeting the constraint goal). Unlike for the constraint part of the intent, the specification of the objective function does not come with any kind of goal to compare against. Thus, the verdict expression compares the behavior of Proteus against that of the oracle.

To enable this comparison, the SUT returns a “cumulative normalized objective function” (CNOF), which aggregates the objective function, after normalizing it to always return a positive value. A high CNOF thus means that the system is behaving well (with respect to the objective function). To take into account the possibility that Proteus’ objective function may exceed that of the oracle, an auxiliary “relative objective function oracle advantage”, computed based on the CNOF of the oracle and Proteus traces, is used in the verdict expression.

Rice has provided the following pseudo-python algorithm and threshold values for the final verdict calculation of CP1 using the above-described fields.

```

1  # Example thresholds, assigned specific values for each challenge problem
2  args.GLOBAL_OBJECTIVE_FUNCTION_ERROR_THRESHOLD = 0.6
3  args.GLOBAL_CONSTRAINT_ERROR_THRESHOLD = 0.15
4
5  # Compute a measure of how much better the oracle's objective
6  # function is, compared to that of Proteus, and set it to 0
7  # when Proteus surpasses oracle.
8  def relative_objective_function_oracle_advantage():
9      fo = cumulative_objective_function(oracle)
10     fp = cumulative_objective_function(proteus)
11     return 0 if fo <= fp else (fo - fp) / max(abs(fo),abs(fp))
12
13 # Compute the verdict (either PASS or FAIL)
14 def verdict():
15
16     if relative_cumulative_absolute_error(proteus) > \
17         args.GLOBAL_CONSTRAINT_ERROR_THRESHOLD:
18
19         return verdict(Verdict.FAIL, "since Proteus does not meet the constraint")
20
21     else: # Proteus does meet the constraint
22
23         if relative_cumulative_absolute_error(oracle) > \
24             args.GLOBAL_CONSTRAINT_ERROR_THRESHOLD:
25
26             # PASS since the oracle does not meet the constraint
27             return Verdict.PASS
28
29         else: # Both Proteus and oracle do meet the constraint
30
31             if abs(relative_cumulative_absolute_error(proteus) - \
32                 relative_cumulative_absolute_error(oracle)) < \
33                 args.GLOBAL_CONSTRAINT_ERROR_THRESHOLD \
34                 and \
35                 relative_objective_function_oracle_advantage() < \
36                 args.GLOBAL_OBJECTIVE_FUNCTION_ERROR_THRESHOLD:
37
38                 # PASS since Proteus comes close enough to oracle in both
39                 # constraint error and objective function
40                 return Verdict.PASS
41
42     else:
43

```

```

44     # FAIL since Proteus does not come close enough to oracle in
45     # both constraint error or objective function
46     return Verdict.FAIL

```

Using the above algorithm, we try to visualize the results through the plots in Figure 6, where all tests are divided into their respective test cases (as described in the Testing Strategy). For each test in each test case, we draw two plots:

1. The bar graph represents, for each of Perturbed (red) and Adapted (green) test stages, a plot of relative cumulative absolute error difference between Proteus and oracle, which must be a value less than solid orange vertical line for a **PASS** verdict.
2. The point plot represents, for each of Perturbed (red triangle) and Adapted (green diamond), a plot of the `relative_objective_function_oracle_advantage`, which must be a value less than the dotted blue vertical line for a **PASS** verdict.

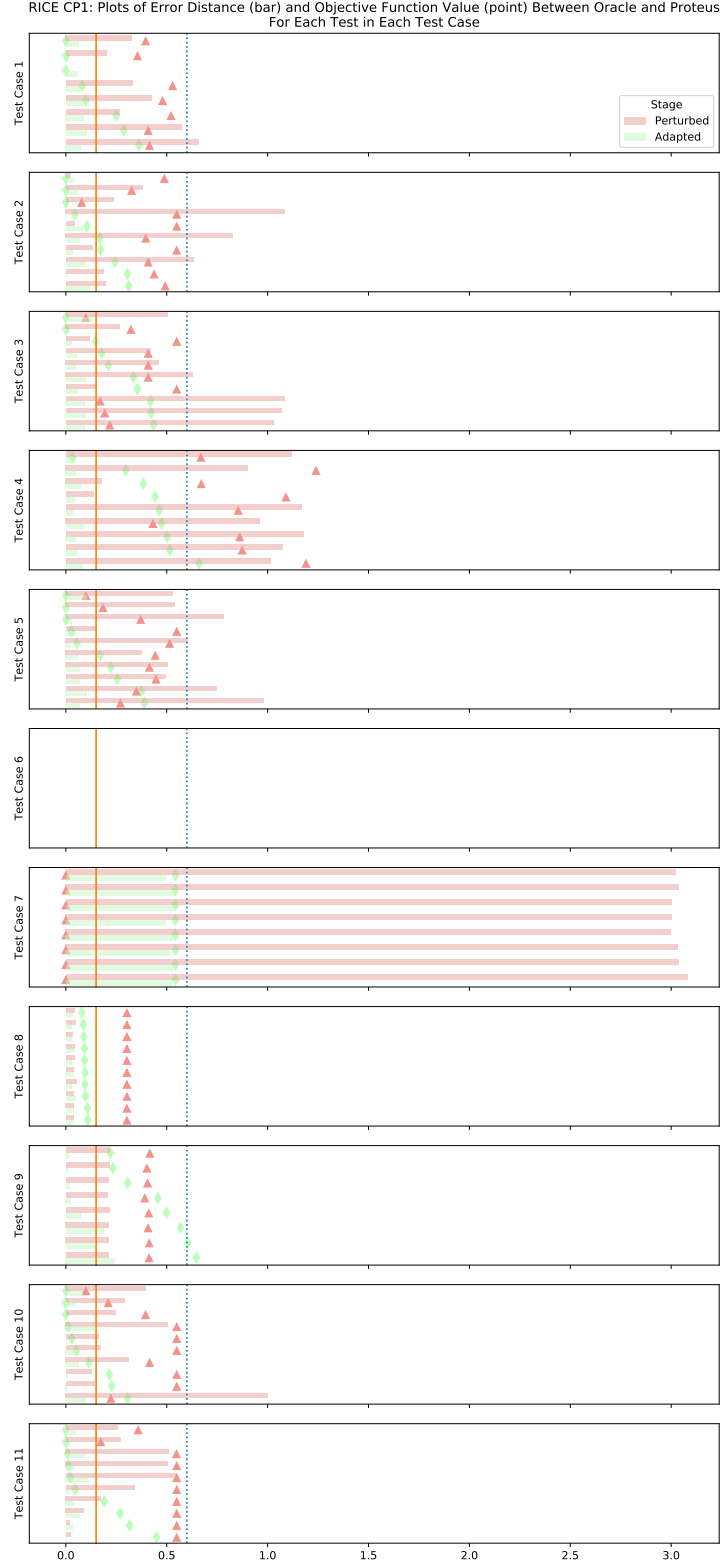


Figure 6. Plots for each test in each test case where the bar graphs are the relative cumulative absolute error difference between Proteus and oracle, and the point plot shows the relative objective function with oracle advantage. Smaller is better. The solid orange vertical line is the upper threshold for PASS for the bar graph and the dotted blue vertical line is the upper threshold for PASS for the point plot.

Note that there are some caveats to the plots that make it inaccurate to draw final verdict conclusions: Specifically, the graph does not indicate if the oracle or Proteus relative cumulative absolute error values are greater than the global constraint error threshold (lines 16-17 and 23-24). Furthermore, due to a misunderstanding and lack of time, test case 6 was not tested, but since it was supposed to fail, it doesn't invalidate the results obtained from the other tests.

Designing an evaluation criterion (verdict expression) for the kind of random tests that Proteus was subjected to during the evaluation requires making a trade-off between false positives and false negatives. The traces of the set of test executions form a distribution, and classifying executions as either **PASS** or **FAIL** requires setting thresholds that determine what class an execution should belong to.

A false negative occurs when a correct execution is marked with a **FAIL**, because variability inherent in e.g. the input stream or in non-functional measures such as energy consumption or performance causes rare, borderline incorrect behavior.

A false positive occurs when an incorrect execution is marked as a **PASS**, because the thresholds for correct behavior have been set too high.

The trade-off then is between making the tests blind to incorrect behavior (reducing false positives), and introducing additional, manual, labor in inspecting the **FAILED** tests, ensuring that they are indeed false negatives. Presenting faulty data as correct to downstream information consumers could impact overall mission success for DoD scenarios, the testing prioritized reduction of false positives. The tests will exhibit a certain amount of false negatives and, looking at the outcome and verdict plots in Figure 7, we indeed see that there were a few **FAILS**. These predominantly correspond to a small number of false negatives, due to inherent variability in the measures that underly the statistics that the verdict is computed from. After further analysis, we found that adjusting the **GLOBAL_CONSTRAINT_ERROR_THRESHOLD** and **LOCAL_ERROR_THRESHOLD** to 0.4 (to match the thresholds used in CP2) produced more accurate results. The team from Rice re-ran the **FAIL** tests and generated Figure 8 with the existing and new thresholds.

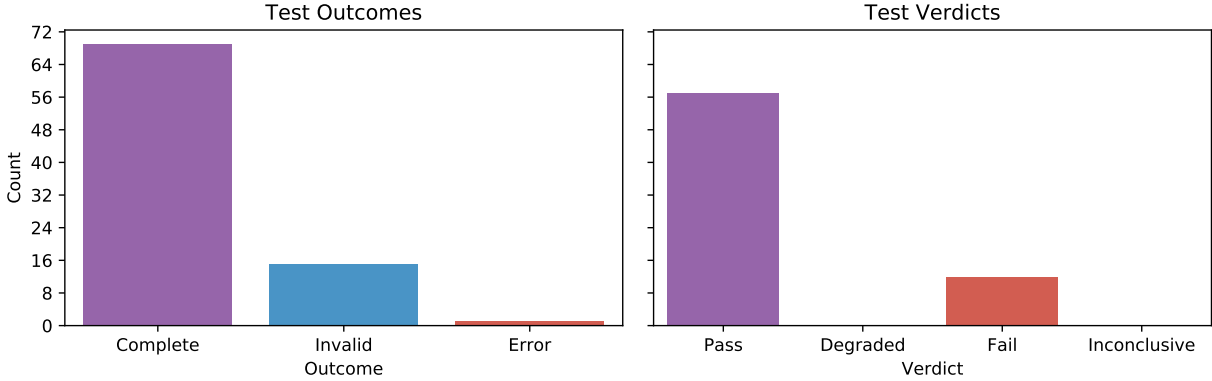


Figure 7. Verdicts and outcomes for CP1 without test case 6 and 7 showing predominantly **PASS** verdicts with a smaller number of false-positive **FAIL** verdicts.

Furthermore, the number of Invalid tests are the result of tests from Test Case 8 which passed in the Perturbed stage, i.e., the system was perturbed but not allowed to adapt and was able to achieve the goals anyway. Normally we would expect failures at this stage to indicate that the test is hard enough to be meaningful. Our interpretation is that this is a characteristic of CP1 where it just isn't as stressed by this test case.

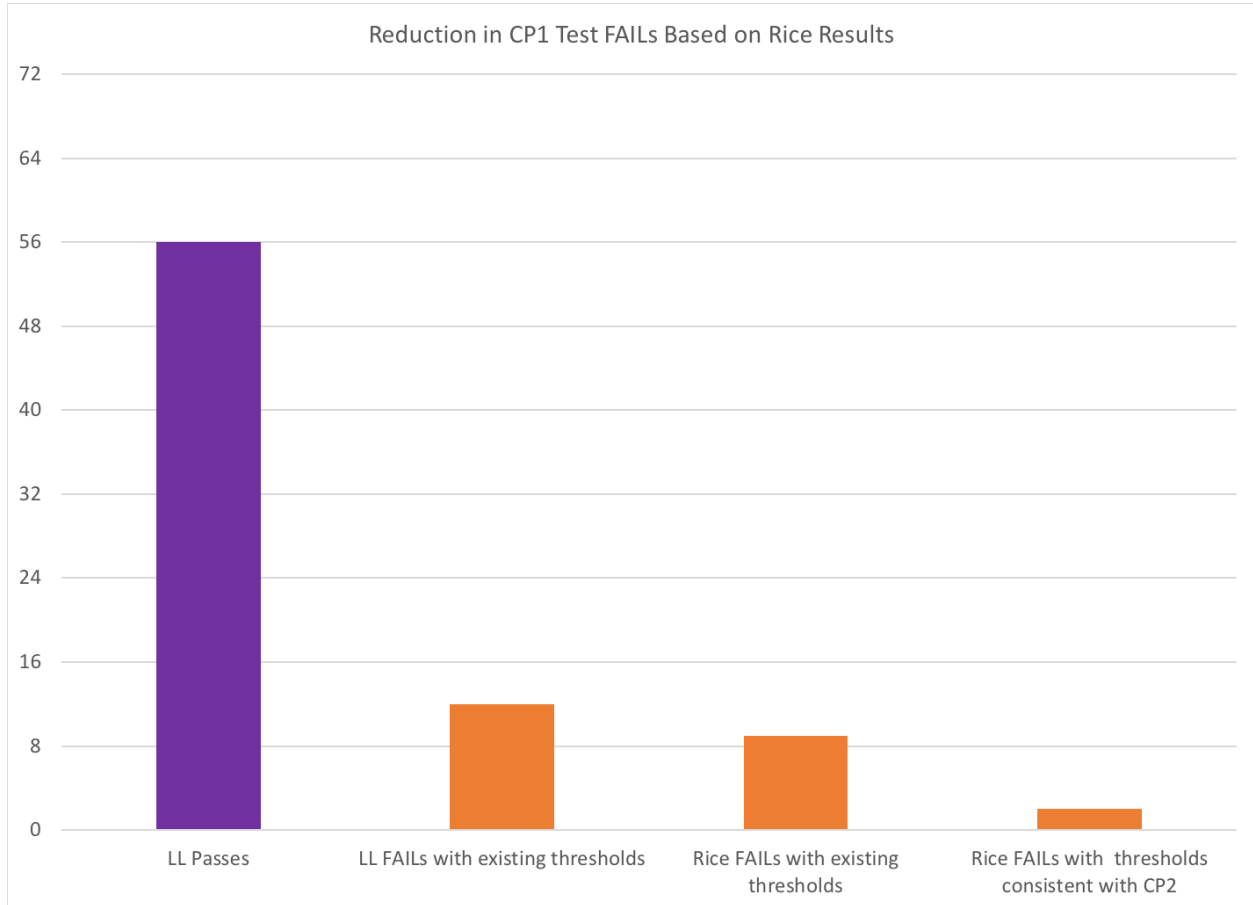


Figure 8. Completed and *FAIL* tests re-run by the Rice team using thresholds run by Lincoln Lab and now compared with thresholds consistent with CP2.

2.2 VIDEO ENCODING (X264)

The second challenge problem represents a streaming video encoding application which was also explored in Phase 1. The mission scenario for this challenge problem is the same as that described in the challenge problem 1 introduction.

Compared to Phase 1, two additional aspects of this scenario are reflected in the challenge problem. The effect of mission length on achievable accuracy is captured by the additional test

parameter `missionLength`. The effect of priority (the “importance” of the scene currently processed by the application) on the quality of the output is captured by the additional test parameter `sceneImportance`. In this Phase the challenge problem focuses on demonstrating that Proteus is able to meet intents by adapting combinations of system and application knobs, a capability which was not demonstrated in the previous Phase. Additional challenges compared to Phase 1 are provided by the presence of an application knob whose values correspond to the use of different algorithms, and a knob that controls an important cost-quality tradeoff in the encoding, and that requires custom instrumentation though more advanced variant of the knob concept.

2.2.1 Testing Strategy

The testing strategy is the same as that described in CP1, however the values used for testing will be relevant to CP2.

2.2.2 Results and Analysis

Figure 9 shows the outcome and verdict plot for CP2. We can see that almost everything passes, and that there are a very small number of what are most likely false negatives, due to threshold limits as described in CP1.

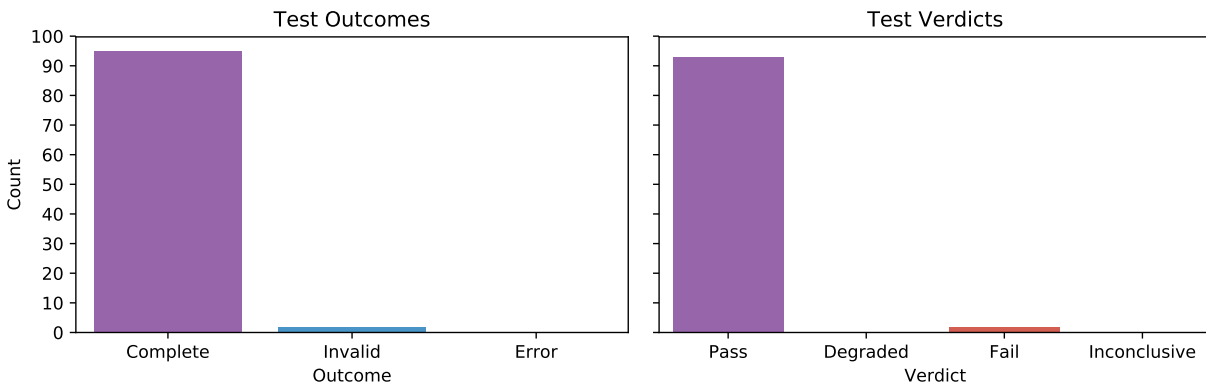


Figure 9. Verdicts and outcomes for CP2 without test case 6 and 7.

2.3 IMAGE MATCHING (CAPSULE)

The third challenge problem represents an image matching problem which is new for Phase 2. The mission scenario for this challenge problem is the same as that described in the challenge problem 1 introduction.

CaPSuLe relies on a compressed database of image hashes, that must be available to the algorithm at runtime, to provide localization based services. The accuracy (exactness of matching) of the algorithm can be improved by increasing the size of the database, meaning that memory usage will likely be a bottleneck for this application in regards to the proposed mission scenario.

The quality metric for this challenge problem is defined as the distance between the minimum and the maximum matching scores reported by CaPSuLe for the images stored in its database.

Like the other two challenge problems this one will also focus on demonstrating that Proteus is able to meet intents by adapting combinations of system and application knobs.

2.3.1 Testing Strategy

The testing strategy is the same as that described in CP1, however the values used for testing will be relevant to CP3.

2.3.2 Results and Analysis

Figure 10 shows the outcome and verdict plot for CP3. We can see that almost everything passes, and that there are a very small number of what are most likely false negatives, due to threshold limits as described in CP1.

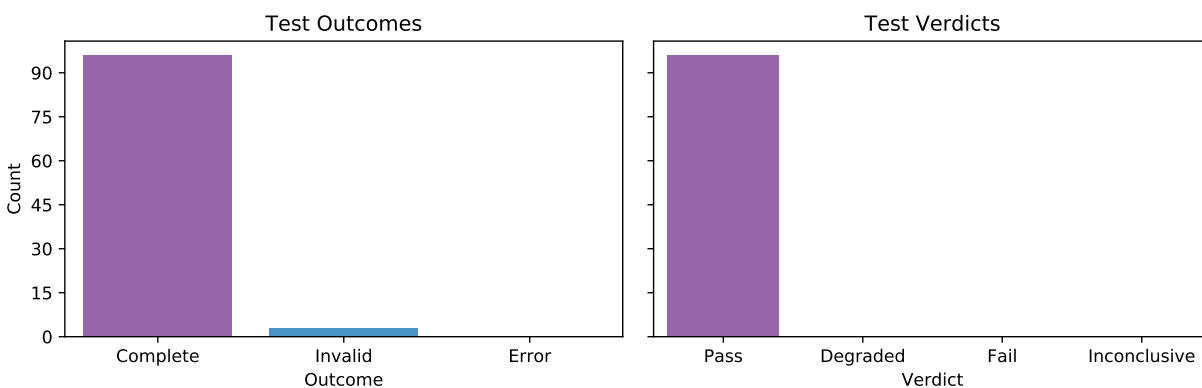


Figure 10. Verdicts and outcomes for CP3 without test case 6 and 7.

REFERENCES

- [1] MIT Lincoln Laboratory, “BRASS Phase 2 Evaluation Methodology,” (Sep 20, 2017), URL https://github.mit.edu/brass/mitll/blob/master/documents/evaluation/evaluation_methodology.md.
- [2] “Proteus Phase II Evaluation Test Plan,” https://github.mit.edu/brass/rice-proteus/blob/c2f2e5d94eaeb13338b5a955c0dcdf28ed8852b6/documents/proteus_phase_2_evaluation_test_plan.pdf, accessed: 08/15/2018.

Delivered to the U.S. Government with Unlimited Rights, as defined in DFARS Part 252.227-7013 or 7014 (Feb 2014). Notwithstanding any copyright notice, U.S. Government rights in this work are defined by DFARS 252.227-7013 or DFARS 252.227-7014 as detailed above. Use of this work other than as specifically authorized by the U.S. Government may violate any copyrights that exist in this work.

Massachusetts Institute of Technology
Lincoln Laboratory

**Building Resource Adaptive Software Systems - Phase 2 Evaluation
SRI: Dynamic Adaptive Embedded Systems (DyAdEm)**

*Timothy Braje, Robert Cunningham, Jeffrey Hughes, Eric Kilmer,
Timothy Meunier, W. Konrad Vesey, Philip Zucker, Martine Kalke*

September 5, 2018

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

This material is based upon work supported by the Defense Advanced Research Projects Agency under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency.

Lexington

Massachusetts

1. EXECUTIVE SUMMARY

The challenge problems provided by SRI are organized around the adaptation needs of autonomous robotic vehicles. Challenge problem 1 attempts to adapt trajectory estimation to the situation of multiple sensor failure. Challenge problem 2 adapts vision feature extraction for different lighting and weather conditions. Challenge problem 3 adapts a global trajectory planner to compensate for missing location markers. All challenge problems used the Robot Operating System (ROS) as infrastructure. In the interaction with the testing harness, the testing harness supplied a perturbation plan at the beginning of the test specifying the time and type of perturbation to apply and then probed for status messages during the run.

1.1 SUMMARY RESULTS AND RECOMMENDATIONS

Each platform and challenge problem is assessed on a number of evaluation criteria which are described in the BRASS Phase 2 Evaluation Methodology [1]. The Phase 2 Evaluation Methodology broadly addresses the general testing strategy as applied to all performers. In the table that follows, we produce a summary of some of the metrics called out in that document.

	CP1	CP2	CP3
Intent Discovery			
Correct	✓	✓	✓-
Complete	✓	✓	✗
Specific	✓	✓	✓-
Fidelity	0.47	1	0
Test Coverage	medium sample	medium sample	medium sample

TABLE 1

Summary statistics for SRI DyAdEm platform.

Summary Results

For CP1, there appears to be a modest performance improvement in path estimation at the edges of the expected range of sensor degradations. For a large majority of that range, it is not entirely clear from the test data available that much adaptation is needed. CP2 passes according to the metrics of the problem. In CP3, the coarseness of returned results makes it inconclusive if the adaptation shows much improvement. For all challenge problems, designing tests with an intent to stress the capabilities of the system would have been more informative and the veil of black box testing makes it difficult to draw strong conclusions about the performance of the systems.

2. DETAILED RESULTS

2.1 SENSOR FAILURE ADAPTATION TO MULTIPLE SENSOR FAILURES

The goal of this challenge problem is to be able to adapt software to handle multiple sensor failures, both transient and/or permanent failures including degraded sensor quality (as was in phase 1, CP1) when multiple (redundant) IMUs and non-redundant GPS, Odometer and vision (LIDAR) sensors are available.

The robotic platform that will be the primary focus of Phase 2 challenge problems is the F1tenth Race Car unlike the Phase 1 which focused on the Segway platform. This platform has higher portability and is easier to use for live demonstrations. The Gazebo simulator is used as a simulation framework and ROS (Robot Operating System) is used as a framework for rapid prototyping of robotic systems.

An additional external IMU sensor is added to the navigation system, acting as a complementary sensor. Having more than one IMU allows the exploitation of sensor redundancy (for IMUs) in the measurements for a variety of tasks, such as fault diagnosis and identification and improving kinematics estimates. The control (navigation) software needs to adapt to a failure in one of the sensors. A failure in one of the aiding sensors (e.g. GPS and Odometer) affects the accuracy of the state estimate, but does not impact the software, whereas a failure in the primary sensor, the low-grade, high-frequency IMU, affects the configuration of the software.

2.1.1 Testing Strategy

The vehicle moves according to a pre-specified linear trajectory. The test harness may specify times at which the parameters of the sensors are degraded. Degradations are selected from plausible ranges of actual vehicle platforms sensors, not necessarily from parameter ranges designed to showcase the recovery of the system.

For each perturbation set, the platform is run in 3 modes of operation. In Baseline, no perturbation is applied. In Perturbed Unadapted, the perturbations are applied but the adaption intended to cope with there perturbations is not turned on. In Perturbed Adapted, the same perturbation is applied but the system is allowed to adapt.

The metrics used for the quality of the adaptation is a comparison of the estimated vehicle path (EVP) to the true vehicle path (TVP). If the adapted system performs well, one anticipates that in the perturbed adapted case the EVP will be closer to the TVP than in the perturbed unadapted case. The Baseline case is used to set an overall scale of expected difference between the EVP and TVP. For more detail, see the challenge problem documentation.

The testing strategy is to perform a broad random sweep of perturbations followed by a closer inspection of possible interesting regions of perturbations parameters.

2.1.2 Results and Analysis

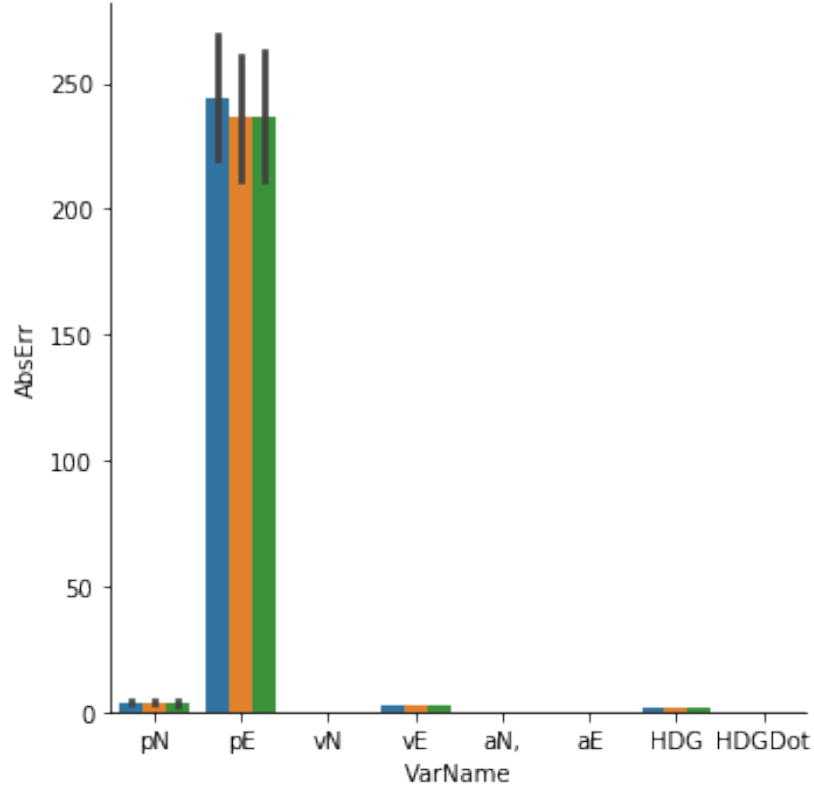


Figure 1. Absolute error of $|EVP - TVP|$ per state variable

With an initial random sweep of perturbations, degradation is not readily apparent. The given range is nominal expected variations of the sensor parameters, which does not appear to cause significant degradation in performance.

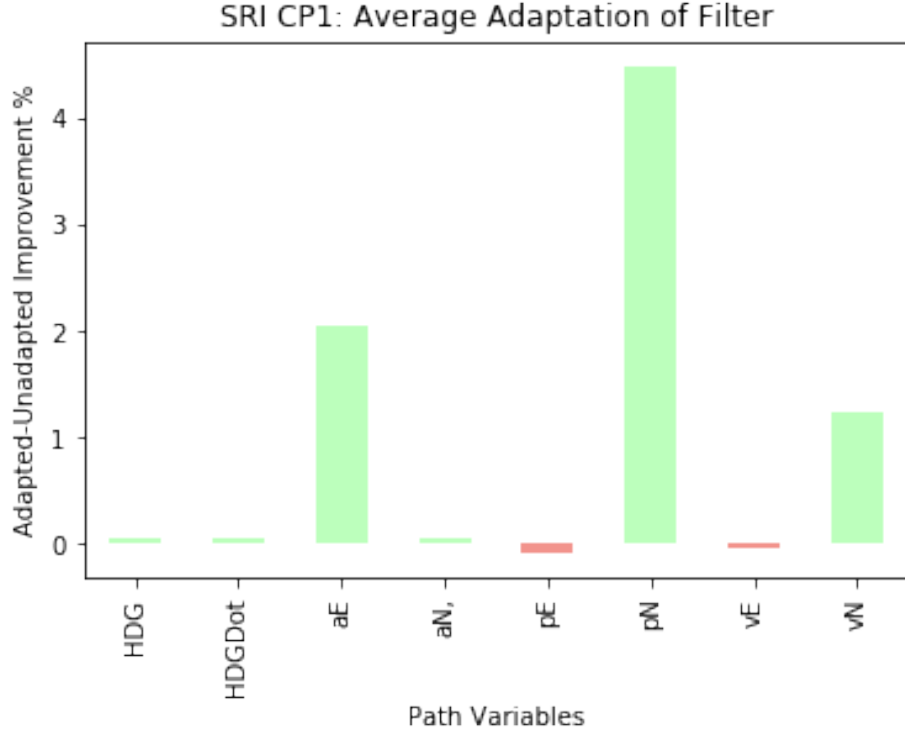


Figure 2. Improvement percentage per variable at the edge of perturbation range

More targeted tests were performed. With perturbations lasting nearly the entire simulation run and with magnitudes at the edges of the specified ranges, degradation is more apparent.

2.2 FEATURE EXTRACTION IN DIFFERENT LIGHTING CONDITIONS

Phase 2, Challenge Problem 2 (CP2) extends the capabilities of the platform by introducing a vision-based feature extraction algorithm for detecting and tracking pertinent features of the environment, which are not known a-priori. The adaptation is designed to improve the performance of the feature extraction algorithm under changing and unpredictable lighting conditions.

As in CP1, the Robotic Operating System (ROS) framework was used for CP2. ROS will allow the easy leveraging of open-source, state-of-the-art vision algorithms to perform the feature extraction (i.e., through OpenCV).

The focus of CP2 is on the effects of lighting changes on the feature extraction algorithms used for downstream applications, such as the vehicle’s navigation filter (as in CP1). Such feature extraction algorithms use a variety of techniques to detect important features of an image (e.g., edges), and track the movement of those features from frame to frame. They typically require

significant tuning, in order to perform near-optimally for a selected environment, and changes in the lighting conditions can dramatically (and detrimentally) affect the performance of the algorithms.

The goal of phase 2 CP2 is to be able to adapt the parameters of a feature extraction algorithm when an unexpected lighting change reduces the effectiveness of the algorithm. In evaluating the effectiveness of the adaption, it is important to evaluate the quality of the features that are extracted under perturbed conditions, as it affects the downstream applications that the features are being used by. As such, the exact metric must capture the amount of information contained in the extracted features, rather than simply the quantity of extracted features.

The Virtual KITTI dataset provides multiple renderings of the same scene, representing the motion of a vehicle on roads, under different lighting conditions; in order to achieve identical scenes, the creators of the data set used artificial computer renderings rather than actual video footage.

2.2.1 Testing Strategy

In order to evaluate the effectiveness of the adaptation, 3 separate runs of the algorithm are required. The test harness selects a sequence from the Virtual KITTI dataset and a time to switch the lighting condition.

As in previous challenge problems, phase 2 CP2 will be evaluated using 3 runs for each scenario.

1. Baseline (B): The baseline run is performed with the unperturbed data set, for which the feature extraction algorithm is already tuned. This will provide a baseline evaluation of how well the feature extraction works for the selected scenario.
2. Perturbed, Unadapted (PU): This run will be performed using the perturbed data set in which the lighting conditions have been changed from the original conditions for which the algorithm was tuned. The feature extraction algorithm will be run with the original parameter set, and the parameters will not change in response to the different lighting conditions.
3. Perturbed, Adapted (PA): This run will be performed using the perturbed data set in which the lighting conditions have been changed from the original conditions for which the algorithm was tuned. The feature extraction algorithm will be run with the original parameter set, but (unlike 2) the adaptation framework will modify the parameters of the feature extraction algorithm, in response to the different lighting conditions.

In each run, the performance of the feature extraction algorithm will be assessed by using a measure known as the Dilution of Precision (DOP). The DOP is a measure of the geometric effects of the extracted features on the navigation solution. It quantifies the quality of the features by computing how useful the feature set is for use in the navigation environment. For a set of features to be highly useful, there must be a sufficient spread of features throughout the geometry of the scene (on both sides, and in front of the vehicle). The DOP is, essentially, the maximum-sized polygonal area that can be constructed from the extracted features.

The PA run performing better than the PU run is considered a success.

2.2.2 Results and Analysis

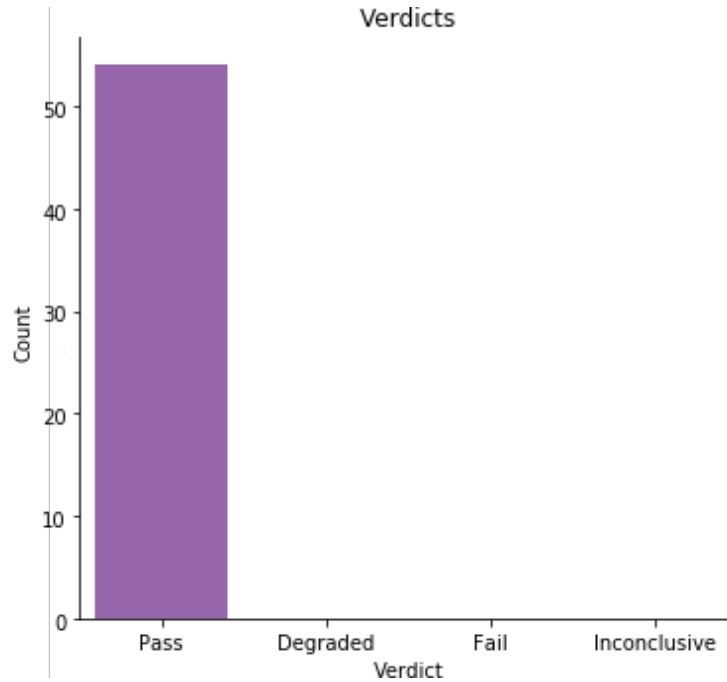


Figure 3. Verdicts of CP2

CP2 appears to pass all tests meaning DOP shows improvement in PU compared to the PA stage.

2.3 ADAPTATION FOR NAVIGATION-AWARE VEHICLE GUIDANCE

Phase 2, Challenge Problem 3 (CP3) employs a 2-level guidance system that includes a global path planner, which computes a feasible trajectory to reach a desired location, and a local planner, which controls the vehicle to follow the global trajectory while avoiding environmental obstacles. The ability of the vehicle to accurately follow its trajectory is closely correlated with the quality of its navigation solution. The quality of the navigation solution, in turn, may be significantly affected by the vehicle's trajectory – the goal of CP3 is to develop a guidance system that adapts its global trajectory, based on local information, to maintain a high level of performance for its navigation solution.

As in CP1 and CP2, the Robotic Operating System (ROS) framework is used and simulations performed in the Gazebo environment. The simulation includes a set of navigation-relevant markers that are distributed randomly throughout the environment. The adaptation in CP3 will change the trajectory of the vehicle, based on the observed markers. The perturbed data set removes a subset of the markers.

2.3.1 Testing Strategy

As an illustration, consider the map shown below. The vehicle (indicated by the blue circle) must move to the goal location (blue star) by following the grid lines (see: fig. a). This requires navigating around a-priori obstacles (red lines) and initially unknown obstacles (yellow lines) that block some routes in the environment (see: fig. b). In addition, the unperturbed map contains a number of unique features (green “+” symbols) that the vehicle can sense when it is near them and its view is unobstructed (see: fig. c). In the perturbed case, some of these markers are removed from the environment (see: fig. d).

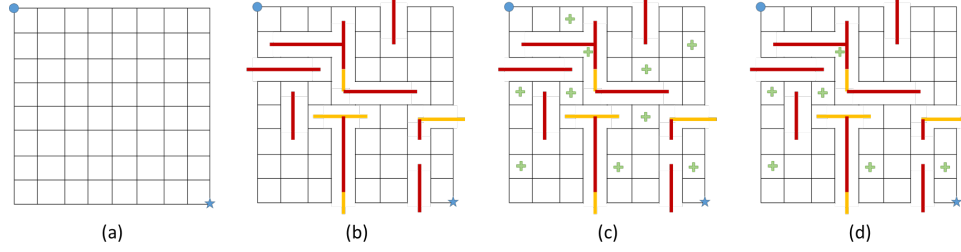


Figure 4. Map

As in previous challenge problems, CP3 will be evaluated using 3 runs for each scenario.

1. Baseline (B): The baseline run is performed with all of the markers are present, for which the guidance algorithms are designed to take the shortest path to the goal location.
2. Perturbed, Unadapted (PU): This run will be performed using the perturbed data set in which a significant portion of the markers are removed from the original map. The guidance algorithms will not be adapted, and will still take the shortest path to the goal location (i.e., the trajectory will be the same as in the baseline run).
3. Perturbed, Adapted (PA): This run will also be performed using the perturbed data set in which a significant portion of the markers are removed from the original map. The guidance algorithms will be run as before, but the adaptation framework will use local information, collected during the run, to modify the parameters and map for the global planner (resulting in changes to the trajectory) in response to changes in the observed markers.

In each run, the performance of the guidance algorithms will be assessed by tracking the number of unique features that are discovered during the execution. To enforce the requirement that the vehicle navigate to the goal location (i.e., rather than simply explore the environment to find as many markers as possible), the perturbation will also specify a time parameter Δt requiring that any change in the vehicle’s plan will increase the duration of the mission by no more than Δt seconds. The performance of any given run can then be viewed as the number of unique markers that were discovered, constrained by the allowable increase in mission duration.

2.3.2 Results and Analysis

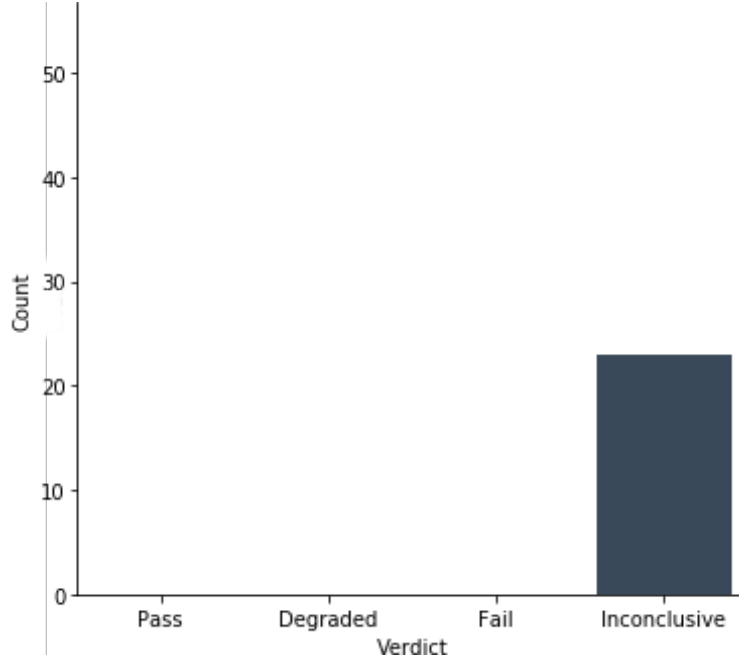


Figure 5. Verdicts of CP3

The results are largely inconclusive. There is a noticeable difference between the B and PU/PA stages, however the PU and PA stages return the same result. The following is an example initialization and result object.

```
{'Map': 1, 'NumberOfMarkers': 22, 'RequiredDistanceToGoal': 0.9376636126983277,
  'LatestRandomRemovalTime': 58.18138635466757, 'SpecificMarkersToRemove': [3],
  'AllowableDurationIncrease': 100,
  'EarliestRandomRemovalTime': 19.971797479359076,
  'SpecificMarkerRemovalTimes': [10], 'NumberOfMarkersToRandomlyRemove': 10}
```

```
{'B': 5, 'd': 0, 'PA': 1, 'PU': 1, 'Verdict': 'INCONCLUSIVE',
  'B_Duration': 651.623, 'PA_Duration': 544.709, 'PU_Duration': 538.954,
  'B_VehiclePosition': [11.556064955001604, 13.534449272272566],
  'PA_VehiclePosition': [11.520710973560503, 13.488083845720212],
  'PU_VehiclePosition': [11.509972076598512, 13.490015173240286]}
```

Conversation with SRI suggests that an increased number of markers (~50-75), removing around half or more of the markers and an early removal time should show results that have a significant number of passing cases.

REFERENCES

- [1] MIT Lincoln Laboratory, “BRASS Phase 2 Evaluation Methodology,” (Sep 20, 2017), URL https://github.mit.edu/brass/mitll/blob/master/documents/evaluation/evaluation_methodology.md.

©2018 Massachusetts Institute of Technology.

Delivered to the U.S. Government with Unlimited Rights, as defined in DFARS Part 252.227-7013 or 7014 (Feb 2014). Notwithstanding any copyright notice, U.S. Government rights in this work are defined by DFARS 252.227-7013 or DFARS 252.227-7014 as detailed above. Use of this work other than as specifically authorized by the U.S. Government may violate any copyrights that exist in this work.

Massachusetts Institute of Technology
Lincoln Laboratory

**Building Resource Adaptive Software Systems - Phase 2 Evaluation
University of Washington: Sandcat**

*Timothy Braje, Robert Cunningham, Jeffrey Hughes, Eric Kilmer,
Timothy Meunier, W. Konrad Vesey, Philip Zucker, Martine Kalke*

September 4, 2018

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

This material is based upon work supported by the Defense Advanced Research Projects Agency under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency.

Lexington

Massachusetts

1. EXECUTIVE SUMMARY

The challenge problems provided by the University of Washington largely centered around code synthesis with some formal guarantees that the synthesized code met some specified qualities. In challenge problem 1, code was transformed from traditional sequential code into code that operated in distributed data processing frameworks like Hadoop MapReduce. In challenge problem 2, there was no code synthesis, but a new image format was created to meet trade-offs in the requested bandwidth - image quality space. For challenge problem 3, filesystems are synthesized to meet some consistency guarantees, regardless of the underlying hardware. These are all offline problems, so tests involve verification that the techniques actually work as intended.

1.1 SUMMARY RESULTS AND RECOMMENDATIONS

Each platform and challenge problem is assessed on a number of evaluation criteria which are described in the BRASS Phase 2 Evaluation Methodology [1]. The Phase 2 Evaluation Methodology broadly addresses the general testing strategy as applied to all performers. In the table that follows, we produce a summary of some of the metrics called out in that document.

	CP1	CP2	CP3
Intent Discovery			
Correct	✓	✓	✓
Complete	✓	✓	✓
Specific	✓-	✓	✓
Fidelity	1.0	1.0	1.0
Test Coverage	complete	complete	complete

TABLE 1

Summary statistics for UW Sandcat platform.

For all three challenge problems, the University of Washington provided us with test scripts that iterated over and exhaustively explored a perturbation space. The verdicts were all successful, lending justification to the claims that these techniques and implementations work. Our recommendations for phase 3 are to design challenge problems which have unexpected, interesting test cases. This could allow the evaluation team to find test cases which uncover unexpected behavior in the systems being developed.

2. DETAILED RESULTS

2.1 CASPER

The goal of this challenge problem is to automatically rewrite originally sequential, non-distributed applications to leverage modern distributed data processing frameworks (e.g. Hadoop MapReduce, Spark, etc.) without manual code modification.

2.1.1 Testing Strategy

The tests were provided to us by the University of Washington, who took care of wrapping up the tests into scripts that could be run within Amazon Web Services AWS and leverage the Hadoop services that AWS provides. These tests are derived from five open source benchmarks, each of which have example applications which contain sequential and manually converted implementations. The Casper system performs its own automatic conversion on the sequential program, and then runs the three versions (sequential, manually converted, automatically converted) against datasets of varying sizes (1, 10, and 100 GB).

For each sample application, intent preservation is verified by demonstrating the following:

- The adaptive version of the application correctly evaluates its associated test datasets, given a dataset size that leads to failure (e.g., out-of-memory errors, timeouts) on the original sequential version of the application.
- The adaptive version of the application achieves some speedup > 1 (on the same hardware configuration) over the original sequential version of the application.
- On a fixed cluster configuration, the adaptive version of the application executes in time competitive with a manually-rewritten version of the application targeting the same framework.

2.1.2 Results and Analysis

Across the five benchmarks, a total of thirty-six applications were run. For all of these applications, the program was able to be rewritten, and its performance was within the desired bounds with respect to the manually converted application. In short, all tests passed.

Test execution times as a function of test are plotted in Figure 1. In all cases, you can see that the sequential version (original program) performs worse than either the manual (hand-written parallel) or auto (synthesized parallel) implementations. The 1 GB plots are the only one which contain results for the sequential implementation, as the larger data sets produce errors for the sequential implementation. It is not until we process 100 GB datasets that we begin to see much variation between the auto and manual implementations. In most cases, we see the two implementations perform similarly with a few of the benchmarks demonstrating better performance for one of them.

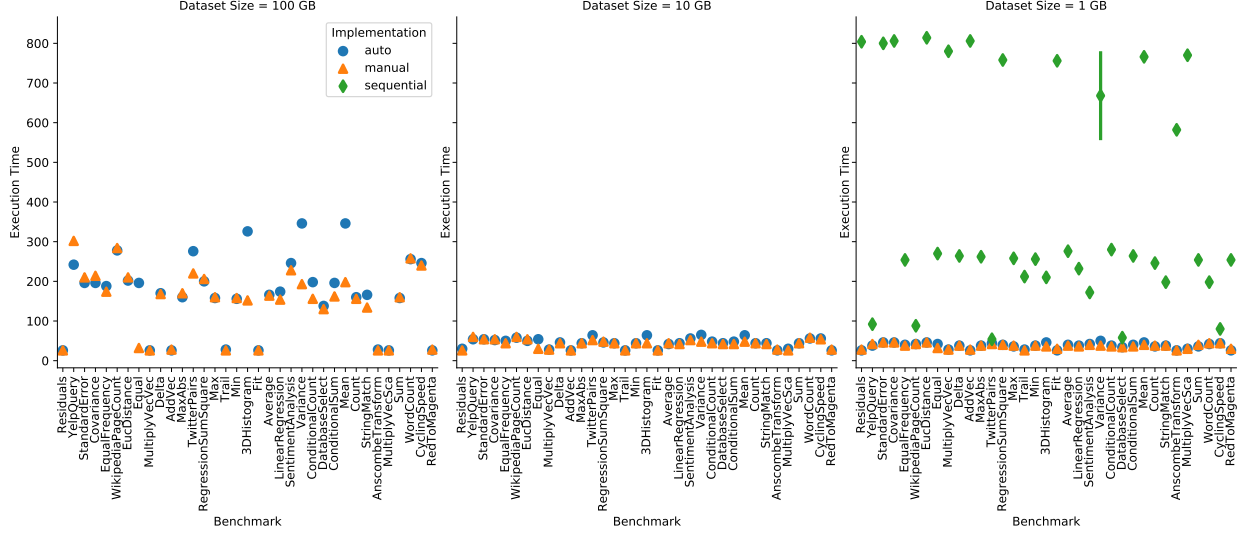


Figure 1. Plot of execution times for each benchmark application.

2.2 A QOS-ADAPTIVE IMAGE STORE

The goal of this test is to compare storage/read bandwidth/quality tradeoffs between the adaptive implementation and the two non-adaptive (baseline) implementations of an online image store which allows one to request images at varying quality levels. The baseline implementations represent two different non-progressive storage techniques which optimize for either bandwidth or storage.

2.2.1 Testing Strategy

The tests were provided by the University of Washington. These tests compared the adaptive implementation vs. the two different baselines for a database of five hundred images and five different quality levels, using a bandwidth-storage capacity product metric.

Preservation of intent (minimizing bandwidth - capacity product) is determined by checking that the adapted image store performs better than both benchmarks, for all image qualities.

2.2.2 Results and Analysis

In Figure 2, we see the bandwidth capacity product metric plotted against different requested qualities for the different systems. We see that the UW adaptive system performs better than the two different baselines for all tested image qualities.

2.3 ADAPTIVE FILE SYSTEMS

The goal of this challenge problem is to empirically validate that the filesystems synthesized via specifications actually meet their stated consistency guarantees.

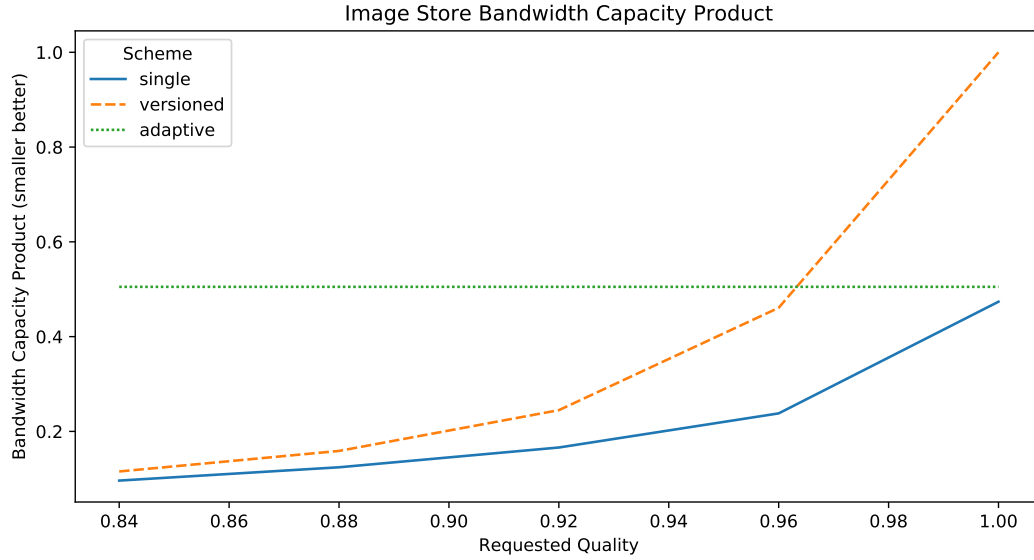


Figure 2. Plot of bandwidth capacity product vs. image quality.

2.3.1 Testing Strategy

The tests were provided by the University of Washington. They work by generating file system images and then executing a variety of verification tests on them, ensuring that they operate as they are supposed to. Success is measured in two ways: 1) correctness, that a new file system achieves the required consistency level; and 2) optimality, that a new file system implementation uses the minimal number of disk barrier operations.

2.3.2 Results and Analysis

The tests (organized into two categories, litmus and verifier) all executed with passing results.

REFERENCES

- [1] MIT Lincoln Laboratory, “BRASS Phase 2 Evaluation Methodology,” (Sep 20, 2017), URL https://github.mit.edu/brass/mitll/blob/master/documents/evaluation/evaluation_methodology.md.

Delivered to the U.S. Government with Unlimited Rights, as defined in DFARS Part 252.227-7013 or 7014 (Feb 2014). Notwithstanding any copyright notice, U.S. Government rights in this work are defined by DFARS 252.227-7013 or DFARS 252.227-7014 as detailed above. Use of this work other than as specifically authorized by the U.S. Government may violate any copyrights that exist in this work.