



Editor: Giuliano Antoniol
Polytechnique Montréal



Editor: Phillip Laplante
Pennsylvania State University



Editor: Steve Counsell
Brunel University

Robotic Testing of Mobile Apps for Truly Black-Box Automation

Ke Mao, Mark Harman, and Yue Jia

ROBOTS ARE WIDELY used for many repetitive tasks. Why not software testing? Robotic testing could give testers a new form of testing that's inherently more black-box than anything witnessed previously. Toward that end, we developed Axiz, a robotic-test generator for mobile apps. Here, we compare our approach with simulation-based test automation, describe scenarios in which robotic testing is beneficial (or even essential), and tell how we applied Axiz to the popular Google Calculator app.

Why Do Robotic Testing?

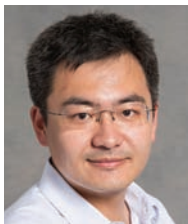
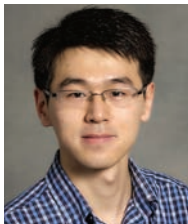
Robotic testing can address the profound shift^{1,2} from desktop to mobile computation. This trend is projected to gather steam,³ accelerated by a concomitant shift from desktop to mobile-device ownership. Automated software testing is needed more than ever in this emerging mobile world. However, we might need to rethink some of the principles of software testing.

Mobile devices enable rich user interaction inputs such as gestures through touchscreens and various signals through sensors (GPS, accelerometers, barometers, near-field communication, and so on). They serve a wide range of users in heterogeneous and dynamic contexts

such as geographical locations and networking infrastructures. To adequately explore and uncover bugs, testing must be able to take into account complex interactions with various sensors under a range of testing contexts. A survey of mobile-app development indicated that practical mobile-app testing currently relies heavily on manual testing, with its inherent inefficiencies and biases.⁴ Frameworks such as Appium (appium.io), Robotium (github.com/RobotiumTech/robotium), and UIAutomator (developer.android.com/topic/libraries/testing-support-library/index.html#UIAutomator) can partly support automatic test execution. However, they rely on human test script design, thereby creating a bottleneck.

Fortunately, many advances in automated Android testing research have recently occurred.⁵⁻⁸ However, these techniques use intrusive (partly or fully white-box) approaches to execute the generated test cases. They also assume that testing tools will enjoy developer-level permissions, which isn't always the case.

Many such techniques need to modify the app code or even the mobile OS, while even the most black-box of approaches communicate with the app under test



(AUT) through a test harness. This isn't truly black-box because it relies on a machine-to-machine interface between the test harness and AUT.

A truly black-box approach would make no assumptions, relying only on the device-level cyber-physical interface between the human and app. Testing at this abstraction level also more closely emulates the experience of real users and thus might yield more realistic test cases. Furthermore, such an approach is inherently device independent, a considerable benefit in situations that might involve more than 2,000 different devices under test.⁹

A Robotic-Testing Manifesto

Handheld devices require rethinking what black-box testing really means. Their user experience is so different from that of desktop applications that existing machine-to-machine black-box test generation lacks the realism, usage context sensitivity, and cross-platform flexibility needed to quickly and cheaply generate actionable test cases.

This section sets out a manifesto for robotic testing in which the generated test cases execute in a truly black-box (entirely nonintrusive) manner. Table 1 compares manual, simulation-based, and robotic testing.

Increased Realism

For Android testing, MonkeyLab generates test cases based on app usage data.¹⁰ Researchers have also published several approaches to generating realistic automated test input for web-based systems.¹¹ However, these automated test-input-based systems don't target mobile platforms, and the overall body of literature on automated test input generation has paid comparatively little attention to test case realism.

A developer won't act on a test sequence that reveals a crash if he or she believes that the sequence is unrealistic. Also, all automated test data generation might suffer from unrealistic tests owing to inadequate domain knowledge. Mobile computing introduces an additional problem: a human simply might not be able to perform the tests. For example, they might require simultaneous clicking with more than five fingers.

In comparison, a robotic test harness can physically simulate human hand gestures. Although there might be some human gestures a robot can't make (and others that a robot can make but no human can replicate), the robotic gestures will at least be physical gestures. As such, those gestures will be closer to true human interaction than the virtual gestures simulated by current nonrobotic test environments, which simply "spit" a generated sequence of events at the AUT.

Device Independence

Existing white-box and (claimed) black-box automated testing requires modifying the behavior of the AUT, the platform, or both. Even techniques regarded as black-box communicate with apps through simulated signals rather than signals triggered through real sensors (for example, touchscreens or gravity sensors) on mobile devices.

As we mentioned before, robotic testing uses the same cyber-physical interface as the human user. It's also less vulnerable to changes in the underlying platform, API interfaces, and implementation details. In a world where time to market is critical, the ability to quickly deploy on different platforms is a considerable advantage.

A Better Cost-Benefit Ratio

Human-based testing is considerably expensive yet enjoys much realism and device independence. In contrast, current automated test data generation is relatively inexpensive, relying only on computation time, yet it lacks realism and device independence. Robotic testing seeks the best cost-benefit ratio and combines the best aspects of human-based testing and machine-to-machine automated testing.

Although robotic technology has historically proven expensive, we're witnessing a rapid decrease in robotic technology's cost. Crowdsourcing, too, is reducing the cost of human-based testing¹² but is unlikely to ultimately be cheaper than robotic testing.

Reduced Reliance on Assumptions

Traditional automated testing makes a number of assumptions about the system under test, whereas human-based test data generation makes fewer assumptions. Robotic testing is much closer to human-based testing in the number of assumptions made, yet its ability to generate large numbers of test cases cheaply is much closer to existing automated testing.

Axiz

Figure 1 shows the Axiz architecture, which contains two high-level components: the robotic-test generator and robotic-test executor.

The Robotic-Test Generator

The robotic-test generator analyzes the AUT and uses the extracted information (including app categories, static strings, and APIs) to adjust a realism model. This model uses previously collected empirical data containing known realistic test cases.

TABLE 1

Criteria to consider when choosing manual, simulation-based, or robotic testing.

Aspects	Considerations	Manual testing	Automated testing		
			Simulation-based	Device-based	Robotic
Target	Test apps	Yes	Yes	Yes	Yes
	Test devices	Limited	No	Limited	Yes
Dependency	Platform support	Cross-platform	Platform-dependent	Platform-dependent	Cross-platform
	Platform version	Independent	Dependent	Dependent	Independent
Integrity	Modify OS	Not needed	In most cases	In most cases	Not needed
Permission	Developer privilege	Not needed	Needed	Needed	Not needed
Cost	Cost	High	Very low	Low	Medium
Scalability	Scalability	Very low	Very high	High	Medium
Interaction	Realism	High	Very low	Low	Medium
	Complexity	Moderate	Very complex	Very complex	Complex
	Sensor activation	Uncontrolled	No	No	Controlled
Performance	Accuracy	Varies	Very high	Very high	High
	Speed	Slow	Fast	Fast	Moderate
	Reliability	Low	High	High	High
User experience	Test imaging	Limited	No	Yes	Yes
	Test touchscreen	Limited	No	No	Yes
	Test inertial measurement unit and near-field-communication sensors	Limited	No	No	Yes
	Test UI response	Limited	Limited	Yes	Yes
Functionality	Oracle	Human	Automated	Automated	Automated
	Internal states	Not accessible	Accessible	Accessible	Not accessible
	Test location-based service	Yes	No	No	Yes
Compatibility	Hardware	Yes	No	Yes	Yes
	Platform	Yes	Limited	Yes	Yes
	Network	Yes	No	Yes	Yes

On the basis of observations of human usage, we compute a comprehensive list of properties (for example, the delay between two adja-

cent events, event types, and event patterns) that capture the underlying real-world test cases' characteristics and properties. We hope these char-

acteristics capture what it is to be realistic, so that Axiz can use them to guide and constrain automated test data generation.

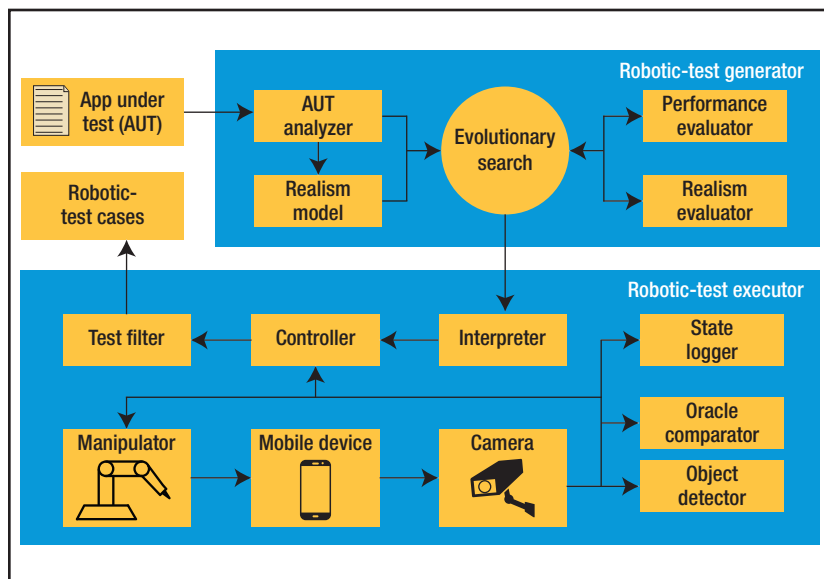


FIGURE 1. The architecture of the Axiz robotic-testing system. The robotic-test generator generates realistic tests. The robotic-test executor filters out unexecutable tests and executes the rest.

The robotic-test generator passes the realism model and AUT to the evolutionary-search component, which generates and evolves test cases. These test cases' realism derives from two aspects of our approach. First, by reusing and extending realistic test cases (for example, Robotium or Appium test scripts), we draw on previous tests manually written by the app testers. Second, by searching a solution space constrained by the realism model, we focus on generating test cases that meet the constraints identified earlier from crowdsourced tests.

We evaluate the generated test cases' fitness on the basis of their performance (such as code coverage and fault revelation) and realism as assessed by the realism model.

The Robotic-Test Executor

We further validate the test case candidates by executing them on a physical device so that they interact with it

in much the same way users or manual testers might do. The robotic-test executor translates the coded test scripts into machine-executable commands for the robot and then executes them on a robotic arm.

The arm interacts with the mobile device nonintrusively, just as a human would. This process requires inverse kinematics and calibration components to make the manipulator act accurately. A camera monitors the mobile-device states. The robotic-test executor further processes image data from a camera through computer vision techniques, which perform object detection and oracle comparison.

Finally, the robotic-test executor sends the overall process data logged during the execution process to the test filter to determine whether the candidate test case is executable in a real-world setting. If not, the executor filters it out. Otherwise, Axiz saves the test for reuse.

A Prototype Implementation

We implemented a prototype of Axiz to demonstrate the system's feasibility (see Figure 2). We built our implementation entirely from commodity hardware components, which are inexpensive, widely available, and interchangeable. We use 3D vision-based self-calibration¹³ to help calibrate and adjust the robotic manipulator to keep the system working reliably and to serve as input to the oracle comparator.

The manipulator is a four-axis Arduino-based robotic arm. It's driven by stepper motors with a position repeatability of 0.2 mm. The maximum speed of movement for each axis ranges from 115 to 210 degrees per second (when loaded with a 200-g load, a sufficient maximum for most mobile devices). At the arm's end is a stylus pen that simulates finger-based gestures.

An external CMOS 1,080-pixel camera monitors the test execution. We run the test generator and robot controller on a MacBook Pro laptop with a 2.3-GHz CPU and 16 Gbytes of RAM.

We employ inverse kinematics (in Python) for robotic-arm control. The object detector and oracle comparator are implemented on top of the OpenCV library. The robotic-test generator employs NSGA-II (Non-dominated Sorting Genetic Algorithm II), a widely used multi-objective genetic algorithm, for multi-objective search-based software testing, using our (currently state-of-the-art) tool Sapienz.⁸ This tool generates sequences of test events that achieve high coverage and fault revelation with minimized test sequence length.

Axiz and the Google Calculator App

The Google Calculator app has had

5 to 10 million installs.¹⁴ Although it's simple, it's a nontrivial real-world app and thus illustrates the potential for truly black-box robotic testing.

We used the robotic-test generator to generate realistic tests, which we executed using the robotic manipulator. The device under test was a Nexus 7 tablet, with normal user permissions and the official Android OS (without modification). For comparison, we introduced another Nexus 7 on which we allowed more traditional intrusive testing. The second Nexus 7 was directly connected to the robot controller on the MacBook. The test tool for it had developer-level privileges and could modify the OS.

Figure 3 illustrates this process. The MacBook's interpreter component translated the event instructions into motion specifications for the robotic-arm controller. That controller then transformed the specifications into joint angle instructions on the basis of inverse kinematics. As Figure 3 shows, the robotic arm touched the buttons on the first Nexus 7 to perform testing. The oracle comparator witnessed each test event. After each step of the test execution, it captured images through the external camera and validated the mobile-GUI states.

Axiz accurately executed each test event specified in the generated robotic-test cases and passed the required oracle checkpoints, faithfully maximizing Sapienz's abilities.

A video of Axiz performing this testing is at www.youtube.com/watch?v=5SjDAQGloXcm. In it, we demonstrate Axiz side by side with a traditional automated-testing tool that doesn't use a robot arm but simply produces a sequence of events. The

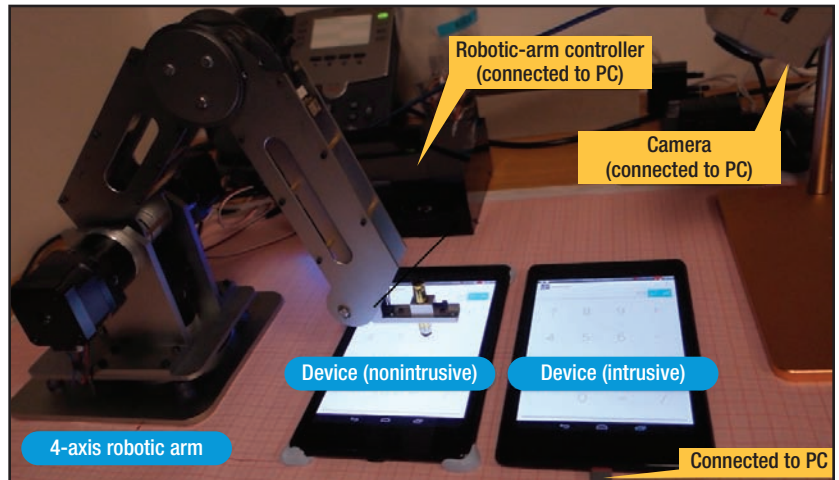



FIGURE 2. Testing mobile apps with a four-axis robotic arm. We built our implementation entirely from commodity hardware components, which are inexpensive, widely available, and interchangeable.

video demonstrates that the robotic arm, built from cheap commodity hardware, can physically produce the same set of events, but more realistically, thereby achieving greater device independence and realism. 

Acknowledgments

We thank Andreas Zeller for his invited talk at the 36th CREST (Centre for Research on Evolution, Search and Testing) Open Workshop,¹⁵ during which he presented a playful video of a disembodied synthetic human hand automatically interacting with a mobile device. This was one of the inspirations for our research.

References

1. F. Richter, "Global PC Sales Fall to Eight-Year Low," 14 Jan. 2016; www.statista.com/chart/4231/global-pc-shipments.
2. "Global Smartphone Shipments Forecast from 2010 to 2019 (in Million Units)"; www.statista.com/statistics/263441/global-smartphone-shipments-forecast.
3. "Worldwide Device Shipments to Grow 1.9 Percent in 2016, While

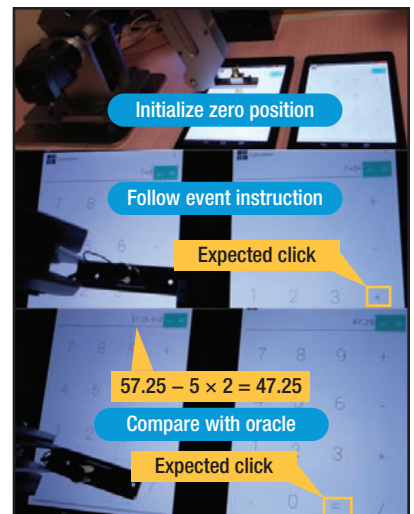


FIGURE 3. Testing a real-world popular calculator app with Axiz. Axiz accurately executed each test event specified in the generated robotic-test cases and passed the required oracle checkpoints.

End-User Spending to Decline for the First Time," Gartner, 20 Jan. 2016; www.gartner.com/newsroom/id/3187134.

4. M.E. Joorabchi, A. Mesbah, and P. Kruchten, "Real Challenges in Mo-

- bile App Development,” *Proc. 2013 ACM/IEEE Int’l Symp. Empirical Software Eng. and Measurement (ESEM 13)*, 2013, pp. 15–24.
5. A. Machiry, R. Tahiliani, and M. Naik, “Dynodroid: An Input Generation System for Android Apps,” *Proc. 9th Joint Meeting Foundations of Software Eng. (ESEC/FSE 13)*, 2013, pp. 224–234.
 6. D. Amalfitano et al., “Using GUI Ripping for Automated Testing of Android Applications,” *Proc. 27th IEEE/ACM Int’l Conf. Automated Software Eng. (ASE 12)*, 2012, pp. 258–261.
 7. W. Choi, G. Necula, and K. Sen, “Guided GUI Testing of Android Apps with Minimal Restart and Approximate Learning,” *Proc. 2013 ACM SIGPLAN Int’l Conf. Object Oriented Programming Systems Languages & Applications (OOPSLA 13)*, 2013, pp. 623–640.
 8. K. Mao, M. Harman, and Y. Jia, “Sapienz: Multi-objective Automated Testing for Android Applications,” *Proc. 25th Int’l Symp. Software Testing and Analysis (ISSTA 16)*, 2016, pp. 94–105.
 9. A. Reversat, “The Mobile Device Lab at the Prineville Data Center,” Facebook, 13 July 2016; code.facebook.com/posts/300815046928882/the-mobile-device-lab-at-the-prineville-data-center.
 10. M. Linares-Vasquez et al., “Mining Android App Usages for Generating Actionable GUI-Based Execution Scenarios,” *Proc. 12th Working Conf. Mining Software Repositories (MSR 15)*, 2015, pp. 111–122.
 11. M. Bozkurt and M. Harman, “Automatically Generating Realistic Test Input from Web Services,” *Proc. IEEE 6th Int’l Symp. Service Oriented System Eng. (SOSE 11)*, 2011, pp. 13–24.
 12. K. Mao et al., “A Survey of the Use of Crowdsourcing in Software Engineering,” *J. Systems and Software*, 2016; [dx.doi.org/10.1016/j.jss.2016.09.015](https://doi.org/10.1016/j.jss.2016.09.015).
 13. J.M.S. Motta, G.C. de Carvalho, and R. McMaster, “Robot Calibration Using a 3D Vision-Based Measurement System with a Single Camera,” *Robotics and Computer-Integrated Manufacturing*, vol. 17, no. 6, 2001, pp. 487–497.
 14. “Calculator,” Google, 2016; play.google.com/store/apps/details?id=com.google.android.calculator.
 15. A. Zeller, “Where Does My Sensitive Data Go? Mining Apps for Abnormal Information Flow,” presentation at 36th CREST Open Workshop (COW 36), 2014; crest.cs.ucl.ac.uk/cow/36/slides/COW36_Zeller.pdf.

KE MAO is a research student at the Centre for Research on Evolution, Search and Testing (CREST) at University College London. Contact him at k.mao@cs.ucl.ac.uk.

MARK HARMAN is the director of the Centre for Research on Evolution, Search and Testing (CREST) at University College London. Contact him at mark.harman@ucl.ac.uk.

YUE JIA is a lecturer of software engineering at the Centre for Research on Evolution, Search and Testing (CREST) at University College London. Contact him at yue.jia@ucl.ac.uk.



This series of in-depth interviews with prominent security experts features Gary McGraw as anchor. *IEEE Security & Privacy* magazine publishes excerpts of the 20-minute conversations in article format each issue.

www.computer.org/silverbullet

*Also available at iTunes

myCS Read your subscriptions through the myCS publications portal at <http://mycs.computer.org>