Robotic Mobile Testing for Truly Black Box Automation

Ke Mao, Mark Harman, Yue Jia
CREST Centre, University College London, UK.

Robots are widely used for many repetitive tasks. Why not software testing? Robotic Testing could give testers a completely new form of 'blackbox' testing, that is inherently *more* black box than anything witnessed previously. This article provides a comparison between the state-of-the-art simulated-based mobile test automation and our proposed robotic mobile testing. We give with scenarios for which robotic testing is beneficial (even essential), introducing a robotic mobile device test generator, Axiz. We illustrate the application of Axiz to a popular Google Calculator app.

Automated Robotic Mobile Testing

We advocate a Ropbotic Tetsing appropach to address the profound shift we are currently witnessing, from desktop to mobile computation[1]. This is a trend which is projected to gather pace[2], accelerated by a concomitant shift from desktop to mobile ownership. Automated software testing is needed more than ever in this emerging mobile world, yet we may need to rethink some of the principles of software testing, even fundamentals, such as what it means to be truly 'black box' when testing.

Mobile devices enable rich user interaction inputs such as gestures via touch screens and various signals via sensors (GPS, accelerometer, barometer, NFC, etc.).

[1] "Global PC sales fall to eight-year low", `http://www.statista.com/chart/4231/global-pc-shipments`; "Global smartphone shipments forecast from 2010 to 2019", `http://www.statista.com/statistics/263441/global-smartphoneshipments-forecast`

[2] "Gartner Announcement", `http://www.gartner.com/newsroom/id/3187134`

They serve a wide range of users in heterogeneous and dynamic contexts such as geographical locations and networking infrastructures. Complex interactions with various sensors under a wide range of testing contexts are required, to adequately explore and uncover bugs.

A recent survey work on mobile app development indicated that current practical mobile app testing relies heavily on manual testing [1], with its inherent inefficiencies and biases. There are frameworks, such as Appium[3], Robotium[4], UIAutomator[5] that can partly support automatic test execution, but they rely on human test script design, thereby creating a bottleneck.

Fortunately, there have been many recent advances in automated Android testing research [2, 3, 4, 5]. However, all of these techniques use intrusive (fully, or partly white box) approaches to execute the generated test cases. They also assume that testing tools will enjoy developer-level permissions; an assumption that does not always hold.

Many such techniques need to modify the app code or even the mobile operating system, while even the most 'black box' of approaches rely on communicating with the app under test through a test harness. This is not 'truly' black box, because it relies on machine-to-machine interface between test harness and app under test.

A *truly black box* approach would make *no* assumptions, relying only upon the device-level cyber-physical interface between human and the app. Testing at this level of abstraction is not only *truly* black box, but it makes fewer assumptions, and more closely emulates the experience of the real user. It may therefore yield more realistic test cases. Furthermore, such truly blackbox approach to testing is inherently device independent; a considerable benefit in the world with more than 2,000 different devices under test[6].

A Manifesto for Robotic Testing

We believe that handheld devices require a rethink of what it means to be black box when testing. The user experience of handheld devices is so different to that for desktop applications, that existing 'machine-to-machine' black box test generation lacks the realism, usage-context sensitivity and cross-patform flexibility needed to quickly and cheaply generate actionable test cases.

This section sets out a manifesto for Robotic Testing, in which the execution of the generated test cases is performed in a truly black box (entirely non-intrusive)

[3]Appium: Automation for iOS and Android apps. `http://appium.io`

[4]Robotium: User scenario testing for Android. `http://github.com/RobotiumTech/robotium`

[5]UIAutomator. `http://developer.android.com/tools/testingsupport-library`

[6]`https://code.facebook.com/posts/300815046928882/the-mobile-device-lab-at-the-prineville-data-center`

manner, using the cyber physical interface of the device, rather than machine-to-machine communication between test two and up on the test[7]. Table 1 compares manual, robotic and traditional automated testing techniques.

Realism: For Android testing, *MonkeyLab* [6] generates test cases based on the app usage data. There are also several published approaches to realistic automated test input generation for web-based systems [7]. Nevertheless, there is little or no attention to realism. A test sequence that reveals a crash, will not be acted upon by a developer who believes the test sequence to be unrealistic. All automated test data generation may suffer from unrealistic tests, because due to inadequate domain knowledge. However, there is an additional problem for the mobile paradigm: the tests may be simply unachievable by human, for example requiring simultaneous clicking using more than five fingers.

Device Independence: Existing white box and (claimed) blackbox automated testing require modification of the behaviour of either the app under test or the platform or both. Even techniques that are regarded as black box, communicate with the app though simulated signals rather than those triggered via real sensors (e.g., touch-screen, gravity-sensor) on the mobile device.

Robotic Testing uses the same cyber-physical interface as the human user, it is 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.

Cost-Benefit: Human based testing is considerably expensive, yet it enjoys a great deal of realism and device independence. By contrast, existing automated test data generation is relatively inexpensive, relying only on computational time, yet it lacks realism and device independence. Robotic Testing seeks the best ratio of cost-to-benefit, and combines the best aspects of human-based testing and existing machine-to-machine automated testing.

Although robotic technology has historically proved to be expensive, we are currently witnessing a rapid decrease in the cost of robotic technology. Crowdsourcing, too, is currently reducing the cost of human-based testing [8], yet it seems unlikely that crowdsourcing would prove to be ultimately cheaper than Robotic Testing.

Reduced Reliance on Assumptions: Traditional automated test techniques make a number of assumptions about the system under test, whereas human-based test data generation relaies on 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

[7]The authors would like to thank Andreas Zeller, for his invited talk at the 36th CREST Open Workshop (COW 36: crest.cs.ucl.ac.uk/cow/36/slides/COW36_Zeller.pdf), at which he gave a playful video of a disembodied synthetic human hand, automatically interacting with a mobile device. This was one of the inspirations for our proposal.

Table 1: *An overview of the criteria to consider when choosing from manual, simulation-based and robotic-based testing approaches.*

Aspects	**Considerations**	**Manual Testing**	**Automated Testing**		
			Simulated Interaction		**Robotic-based**
			Emulator-based	**Device-based**	
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
Scalibility	Scalibility	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	Vary	Very high	Very high	High
	Speed	Slow	Fast	Fast	Moderate
	Reliability	Low	High	High	High
User Experience	Test imaging	Limited	No	Yes	Yes
	Test touch screen	Limited	No	No	Yes
	Test IMU, NFC 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 LBS	Yes	No	No	Yes
Compatibility	Hardware	Yes	No	Yes	Yes
	Platform	Yes	Limited	Yes	Yes
	Network	Yes	No	Yes	Yes

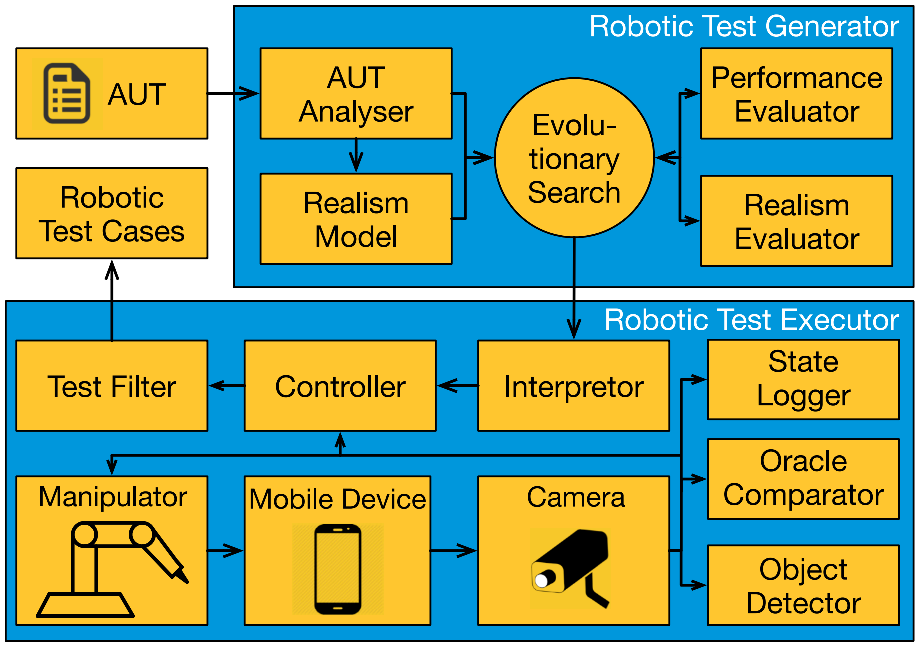

Figure 1: *The framework of the Axiz robotic mobile testing system.*

automated testing.

The Axiz Framework

Our Axiz Robotic Testing system architecture is depicted in Figure 1. The framework contains two high-level components: the 'robotic test generator' (for generating realistic test cases), and the 'robotic test executor' (for further execution and filtering of the tests). The filtering stage removes tests that can be detected to be unexecutable in a real-world setting.

Robotic test generator. The robotic test generator starts by analysing the application under test (AUT). The extracted information of the AUT (including app categories, static strings and APIs, etc.) is used to adjust a 'realism model'. The realism model uses previously-collected empirical data containing known-realistic test cases.

Based on a series of observations of human usage, we compute a comprehensive list of properties (e.g., delay between two adjacent events, event types and event patterns) are calculated to capture the characteristics and properties of the underlying the real-world test cases. The hope is that these characteristics will capture what it is to be 'realistic', so that these can be used to guide and constrain automated test data generation.

The realism model, together with the AUT, are passed into the evolutionary

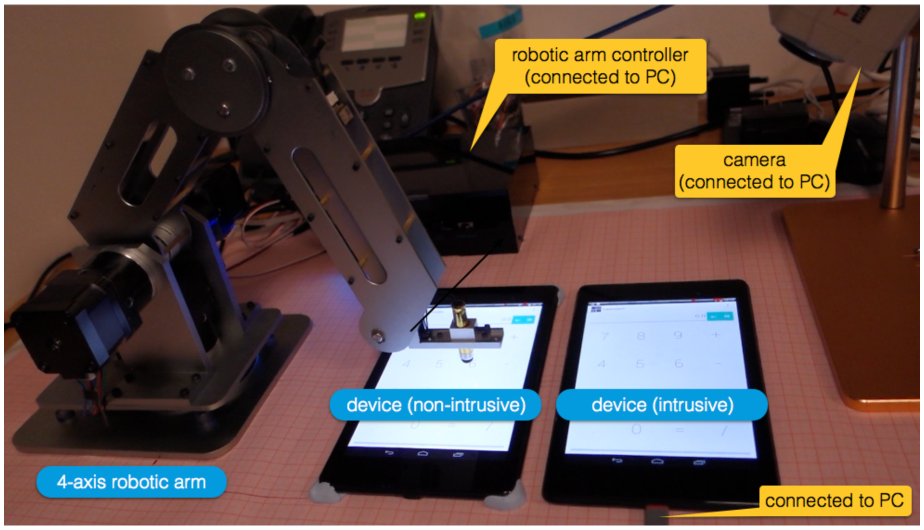

Figure 2: *Testing mobile apps with a 4-axis robotic arm.*

search component for generating and evolving test cases. The source of 'realism' for the individuals being evolved, derives from two aspects of our approach: Firstly, by reusing and extending realistic test cases (e.g., Robotium or Appium test scripts), we draw on previous tests manually written by the app testers. Secondly, by searching a solution space constrained by the realism model, we constrain our search to generate test cases that meet the constraints identified earlier from crowdsourced tests.

The fitness of the generated test cases is evaluated based on their performance (such as code coverage and fault revelation) and realism as assessed by the realism model.

Robotic test executor. The generated test case candidates are further validated by executing them on a physical device, thereby interacting with the device in much the same way that end-users or manual testers might do. The test executor first translates the coded test scripts into machine-executable commands for the robot, and executes them on a robotic arm.

The arm interacts with the mobile device non-intrusively, just as a human would. This process requires inverse kinematics and calibration components in order to make the manipulator act accurately. A camera is used to monitor the mobile device states. Image data from the camera is further processed via computer vision techniques, which perform object detection and test oracle comparison.

The overall process data logged in the execution process is finally sent to a test filter To determine whether the candidate test case should be filtered out.

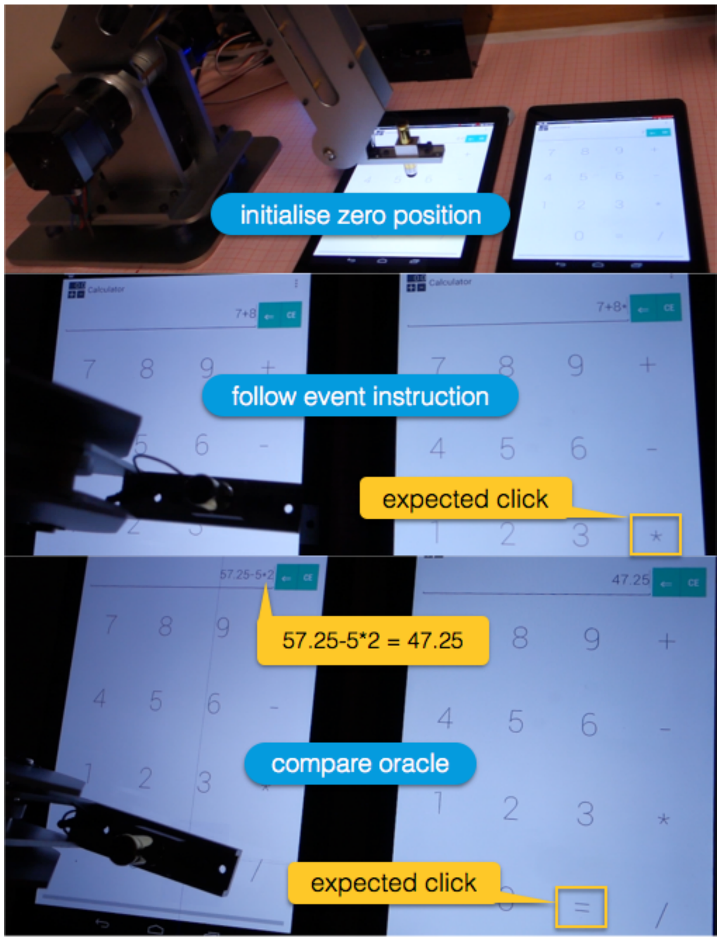

Figure 3: *Testing a real-world popular calculator app with Axiz*[9].

A Prototype Axiz Implementation

We have implemented a prototype of our Axiz system, shown in Figure 2, to demonstrate practical current feasibility. Our implementation has been built entirely using commodity hardware components, which are inexpensive, widely available and interchangeable. We use 3D vision-based self-calibration approach [9] to help calibrate and adjusting the robotic manipulator, in order to keep the system working reliably and as input to the oracle.

More specifically, we use a 4-axis Arduino-based robotic arm as the manipulator. The arm is driven by stepper motors with a position repeatability of 0.2mm. The maximum speed-of-movement for each axis ranges from 115 to 210 degrees per second (when loaded with a 200g load, a sufficient maximum for most current mobile devices). At the end of robotic forearm, a 'stylus pen' is installed to simulate a human's finger-based gestures.

We use a Nexus 7 tablet as the device under test, with normal user permissions and the official Android system (without modification) as the platform and operating system. An external CMOS 1,080p camera is used to monitor the test execution. We run the test generator and robot controller on a MacBook Pro laptop with a 2.3 GHz CPU and 16G RAM.

We implemented inverse kinematics (in Python) for robotic arm control. We implemented the object detector and oracle comparator on top of the on OpenCV library. The test generation component implements a widely used multi-objective genetic algorithm called NSGA-II for multi-objective search based software testing, using our (currently state-of-the-art [5]) tool Sapienz, for generating sequences of test events that achieve high coverage and fault revelation with minimised test sequence length.

An Proof-of-Concept Illustrative Example

We selected the popular Google Calculator app as our sample subject, which has 5 to 10 million installs according to Google Play[8]. Although this is a simple app, it is a nontrivial real-world app and therefore serves to illustrate the potential for truly black box Robotic Testing.

To demonstrate the usefulness of Axiz, we first use Axiz' robotic test generator to generate realistic tests, which we then execute using the Axiz robotic manipulator. For a comparison purposes, in our demonstration, we also introduce another Nexus 7 tablet (for which more traditional intrusive testing is permitted). This comparator Nexus 7 is directly connected to the PC controller. We allow the test tool for it to have developer-level privileges and permit it to perform OS modifications.

An illustration of this process is shown in Figure 3: The interpretor component on the PC translates the event instructions into motion specifications for Axiz' robotic arm controller, which transforms these into joint angle instructions, based on inverse kinematics. As shown in the screenshot, the robotic arm touches the buttons shown in the left-hand device for robotic test execution. The oracle comparison component 'witnesses' each test event; after each step of the test execution, it captures images via the external camera and validates mobile GUI states. A demo video of Axiz is available at: `https://www.youtube.com/watch?v=5SjDAQGloXcm` which shows that Axiz was able to accurately execute each test event specified in the generated robotic test cases and to pass all required oracle checkpoints, faithfully maximising the abilities of our traditional machine-to-machine 'blackbox' tester, Sapienz [5].

[8]https://play.google.com/store/apps/details?id=com.google.android.calculator

References

[1] M. E. Joorabchi, A. Mesbah, and P. Kruchten, “Real challenges in mobile app development,” in *Proc. of ESEM’13*, pp. 15–24, 2013.

[2] A. Machiry, R. Tahiliani, and M. Naik, “Dynodroid: An input generation system for Android apps,” in *Proc. of ESEC/FSE’13*, pp. 224–234, 2013.

[3] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon, “Using GUI ripping for automated testing of Android applications,” in *Proc. of ASE’12*, pp. 258–261, 2012.

[4] W. Choi, G. Necula, and K. Sen, “Guided GUI testing of android apps with minimal restart and approximate learning,” in *Proc. of OOPSLA’13*, pp. 623–640, 2013.

[5] K. Mao, M. Harman, and Y. Jia, “Sapienz: Multi-objective automated testing for Android applications,” in *Proc. of ISSTA’16*, pp. 94–105, 2016.

[6] M. Linares-Vásquez, M. White, C. Bernal-Cárdenas, K. Moran, and D. Poshyvanyk, “Mining Android app usages for generating actionable GUI-based execution scenarios,” in *Proc. of MSR’15*, 2015.

[7] M. Bozkurt and M. Harman, “Automatically generating realistic test input from web services,” in *Proc. of SOSE’11*, pp. 13–24, 2011.

[8] K. Mao, L. Capra, M. Harman, and Y. Jia, “A survey of the use of crowdsourcing in software engineering,” *Journal of Systems and Software*, 2016. `http://dx.doi.org/10.1016/j.jss.2016.09.015`.

[9] 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, pp. 487 – 497, 2001.