© 2017 IJEDR | Volume 5, Issue 3 | ISSN: 2321-9939

Automated Test Oracles for Android Devices to get the Accuracy, Efficiency, and Reusability

D.Anand Kumar, Ramya Paruchuri
M.Tech Scholor, Asst. Lecturer
Computer Science and technology,
Velagapudi Ramakrishna Siddhartha Engineering College, Vijayawada, India

---

Abstract—**Automated GUI testing tool works on simulating user events and validating the changes in the GUI so that an Android application works properly. When testing an application and if the device under test (DUT) is under heavy load the accuracy may degrades significantly. In order to improve the accuracy, our previous work, SPAG, uses event batching and smart wait function to eliminate the uncertainty of the replay process and adopts GUI layout information to verify the testing results. The previous work SPAG outperforms the existing methods with 99.5 percent accuracy when testing. Present, the work is of testing and android device with an extension of SPAG, i.e. Smart phone Smart Phone Automated GUI testing tool with Camera (SPAG-C). The work's goal is to reduce the time required to record the test cases and increase reusability without compromising the test accuracy. SPAG-C automatically performs image comparison with the help of external camera photos instead of frame buffer screenshots to verify the results. In order to make SPAG-C reusable for different devices and to allow better synchronization at the time of capturing images, we develop a new architecture that uses an external camera and Web services to decouple the test oracle. SPAG-C is 50 to 75 percent faster than SPAG in achieving the same test accuracy. With reusability, SPAG-C reduces the testing time from days to hours for heterogeneous devices.**

IndexTerms—**Reusability, testing tools, GUI Testing, SPAG, DUT, Record and Replay**

I. INTRODUCTION

AUTOMATED graphical user interface (GUI) testing tools aim to test graphical user interfaces reduces the manual work done by the testers and use testing methodologies. The two fundamental tasks in automated GUI testing is first, simulating user events, and second, verifying that the application behaves as expected. More specifically, an automated testing tool executes a given set of tests on an application under test (AUT) and verifies its behavior using a test oracle. The test cases have all the information required to simulate user events on the AUT, while test oracles have the mechanisms to capture the current state of the GUI during the testing process and to compare it with the corresponding expected state, which is usually given before the execution of the test case.

To make the GUI testing tools available for different users, three major issues must be addressed: Reusability, Efficiency, and Accuracy. For the reusability most of the times GUI testers would run a test case under several kinds conditions and on different devices to check how the AUT responds; thus the degree of reusability of test cases and testing tools are crucial. Moreover reducing the testing time is the main goal of automated GUI testing tools. This can be carried out by routinely walking test cases and robotically verifying GUI states. Finally, an automated GUI checking out device ought to be capable of as it should be tell whether or not the AUT is behaving as anticipated or now not. It means that a low percentage of false positives and false negatives is desirable.

According to our previous work SPAG, the accuracy of testing tools drops significantly when the device under test (DUT) is heavily loaded, such as running many background processes, transferring data via the Internet and having many concurrently running applications. An application experiencing delay may fail to process an event correctly if the response to the previous event has not been completed. For example, an event may be dropped if the application receives the event ahead of time and is not ready to process it. The dropped event would cause the testing tool to report a false negative.

Improving the accuracy of matching GUI images, however, often conflicts with reusability of test oracles because testing the same application in different devices with different screen sizes makes it impossible to use the same images as oracles in both devices even when testing the same functionality. In order to use the same test cases repeatedly, we must check for similarity in the output, but not exact matches. Checking for similarity, we will neglect some minor errors in matching. Therefore, there is a tradeoff between the accuracy of matching GUI images and the reusability of test oracles. Several methods have been proposed to address the issues of accuracy, efficiency, and reusability.

Different automation approaches address these issues in different ways. For example, "model-based" testing [4], [5] aims to automate as much work as possible by automatically generating test cases and verifying the GUI state. However, the great amount of possible combinations regarding which actions can be performed on a GUI means that this process may take days, weeks, even months to fully test an application depending on how complex its GUI is. Using "accessibility technologies" to get programming access to GUI objects of the AUT is another approach that has been suggested recently. Although this method also works for black box testing, it is limited by many factors, such as the system's API, security restrictions, and the information made available

by developers of the application through accessibility technologies. Finally, another commonly-used automation technique is "record-replay"[6], [8]. This technique allows testers to "record" test cases without writing codes, but by performing GUI actions directly on the application, while a tool automatically creates the test case. Afterwards, testers can "replay" these test cases any number of times. Nevertheless, the record-replay technique still requires testers to record the test cases and provide the expected states to verify the GUI.

One related record-replay work is our previous work shrewd telephone automatic GUI checking out software (SPAG) . SPAG combines Sikuli [2] [9] (an automation tool for computers that makes extensive use of computer vision procedures) and Android Screencast (a faraway control instrument for Android devices) to perform automatic GUI trying out on Android devices. Because both instruments, Sikuli and Android Screencast, are open-supply, SPAG improves them with the aid of including some performance that helps additional automate the trying out approach on Android devices. First, SPAG monitors the CPU utilization of target utility at runtime. Subsequent, SPAG dynamically changes the timing of the following operation so that every one occasion sequences and verification can be performed on time. Compared to existing methods, SPAG continues an accuracy of up to ninety nine.5 percentage and outperforms present approaches. This work (SPAG-C) is an extension of SPAG. It goals to beef up SPAG through bettering checking out effectivity and growing reusability without compromising accuracy.

With a view to reap our objectives, we increase a further architecture. Normally, testing instruments are platform-dependent despite the fact that the verification method is the same whatever the device under test, which means the test Oracle [2] could no longer be reusable. Regularly, there were two ways to confirm the state of a software's GUI: image evaluation and object identification. This work uses image assessment because it permits us to swiftly verify a software's GUI while not having the supply code (black box testing) and it's approach independent, which allows for us to design an procedure that can be utilized in a wider variety of gadgets. But, as an alternative of shooting the required screenshots from within the device as is quite often carried out, we use an outside digital camera. Utilising an outside digicam makes the verification component platform unbiased and offloads some processing from the DUT. Moreover, we use web provider technologies to show the verification factor to the report-replay factor.

It means that the experiment oracle is just not simplest platform unbiased but in addition independent from the record-replay element seeing that now it is accessed by way of internet offerings, because of this it may be accessed by using extraordinary trying out instruments thanks to the interoperability offered by using web provider necessities. Subsequently, we endorse a system to automatically derive the expected states of an utilities GUI in the course of the file approach, which reduces the time required to file test cases.

II EXISTING SYSTEM

SPAG

This work, SPAG-C, is an extension of a previous work called smart phone automated GUI testing tool. SPAG combines and extends two open source tools: Sikuli and Android Screencast [10]. SPAG merges these two tools together to enable using Sikuli's API for testing Android devices. SPAG intercepts user interactions with Android Screencast, saves these interactions in a Sikuli test file and replays them later as required by the tester. SPAG provides three contributions: (1) Batch event, which accurately reproduces the recorded event sequences; (2) Smart wait, which automatically establishes a delay between events to ensure that the DUT has enough time to process previous events; and (3) an automatic verification method, which makes use of Android accessibility services to record transition between activities after an event is executed.

Since SPAG is integrated with Sikuli, it can also take advantage of Sikuli's API to perform image verification in a semi automatic way, which means that the verification is done by Sikuli but the tester still needs to provide the images and to write the commands into the test case. SPAG also provides an automatic verification that uses Android Accessibility Services to gather the name of the activities, and performs a string comparison to verify that the same activity transition that occurred after the input of a specific event during record also happens during replay.

This, however, does not ensure that applications are being displayed as expected. SPAG-C also provides two verification approaches: semi-automatic and automatic. In both approaches SPAG-C performs image verification with images captured from a camera, the only difference is that the semi-automatic approach requires testers to capture the images, while automatic approach does not. SPAG depends on Android Screencast to interact with the DUT; therefore, it inherits its limitations such as limited support for devices, slow response time that affects the image verification process, and the inability to reproduce multi-touch events. Since SPAG-C is based on SPAG, it also inherits some of SPAG's limitations; but we improve the verification process by making it more reusable, automated, better synchronized, and platform-independent.

Image Comparision

Image comparision is the technique used in Verification process of testing tools. Here are some Image Comparision methods used in SPAG and SPAG-C.

Histogram. : It represents the color representation of the image within each pixel or in a group of pixels like we GOOGLE uses in image search. Here it compares two images of color histogram [11]. If the two images similar by histogram, then the images are to be similar. It may vary according to the lighting conditions and to be precise, it has to be with more pixels. If not, different images are considered to be similar images.

SURF (Speeded up robust features). Is a "scale-and rotationinvariant interest point detector and descriptor" [12]. In computer vision, an interest point detector is used to detect parts of an image that can be used to uniquely describe it. An interest point, also called feature or key point, has many properties; perhaps the most important one is its repeatability, which means that it could be reliably computed under different conditions (e.g., changes in size, rotation, etc.). After an interest point of an image has been identified, the interest point descriptor uses the neighborhood information of the interest point to characterize it. By

adopting the characteristics of interest points, SURF-based image comparison method first extracts the interest points of the two images being compared. It then matches descriptors of both images. Finally, image similarity is measured according to the amount of matches.

Template matching. Is a method used to find a small image (template) in a larger image (source). This is done by taking the template and sliding it on the original image pixel by pixel; at every point a metric is calculated to determine how good the match is. After all metrics are calculated, the best match can be selected. Depending on the method used, the best match may be the highest or the lowest calculated value [13].

Other Android based testing tools

Monkeyrunner [15]. Is a testing tool provided by Google. It provides an API that developers can use to control Android devices without the need of any source code. To use Monkeyrunner, developers write Python programs to simulate user interaction. If they want to corroborate the state of the GUI, they can also write commands to capture screenshots from within the devices using Android's frame buffer which is the part of video memory containing the current video frame. There are three main issues with Monkeyrunner apart from the fact that in order to use it testers need programming skills: first, the na€ıve form in which it simulates events on the AUT [7]; second, its verification approach; third, capturing screenshots from Android's frame buffer is time-sensitive, which means that testers need to adequately synchronize the simulation of events with the time of the capture, otherwise invalid images will be taken for verification. On the contrary, SPAG-C takes advantage of the method used by SPAG to accurately simulate events on the DUT, and uses a non-intrusive method to capture images which is automatically synchronized with the simulated events at all times.

Robotium framework [16]. Is a framework used to perform black box testing on Android devices. It uses Android Instrumentation [18] to interact with an application's GUI and gather information. In order to check the state of an application, screenshots can be taken or object identification can be performed using Robotium's API and JUnit's assertions. Robotium is widely used but just like Monkeyrunner, it requires testers to manually program test cases. SPAG-C automatically creates test cases by listening to user events and recording them in the test case which reduces the test writing time considerably.

Testdroid Is an Android testing platform that uses the Robotium framework to define test cases. Testdroid records user interactions and automatically generates Java code with calls to Robotium API. These test cases can be later replayed at any time in the same way that Robotium tests are executed. With Testdroid, testers can execute their tests either locally, on their own devices, or remotely, using Testdroid's cloud services. Testdroid's cloud services provide log files and statistics about test execution; additionally, it takes screenshots during the testing process so developers can verify the GUI. Testdroid services, however, are quite expensive, and GUI verification has to be done manually by the testers since Testdroid does not perform any comparison against expected states. On the contrary, SPAG-C completely automates the verification process so that testers only need to record the tests.

GUITAR Android graphical user interface testing framework (GUITAR) [17] was an effort of Xie and Memon to migrate their previous work [4] on model-based testing to the Android platform. GUITAR consists of two modules: ripper and replayer. The ripper is in charge of automatically generating event-flow graphs for their later conversion into test cases. The ripper does this by automatically interacting with an application and gathering all relevant information about its GUI. Since the GUI ripper cannot be guaranteed to have access to all different windows and widgets of an application, a capture/replay tool was created for testers to complement the ripper. The replayer is in charge of the execution of the generated test cases. A main problem with GUITAR is that it may not be entirely practical on production- ready devices because it uses Hierarchy Viewer, a tool that can only connect to devices running a developer version of the Android system. In addition, GUITAR is platform- dependent even though the verification process is the same regardless of the device under test, which means the test oracle might not be reusable. On the other hand, SPAGC can be used on a great variety of real devices. We use Web service technologies to expose the verification component to the record-replay component. Our test oracle is not only platform-independent but also independent from the record-replay component. Amalfitano et al. discussed a similar problem as GUITAR did. However, no results were shown about the precision of the system when verifying the GUI. In addition, it may take a considerable amount of time to gather the information required to begin testing, and to perform the verification because the crawler needs to go throughout all possible event sequences and all windows. Further, they did not address the problem of event synchronization. If the testing process introduces overhead to the devise and it takes longer for the application to respond, it is not clear whether all the input events can be executed at the right time or not. Finally, their method cannot be used to perform black-box testing, because they instrumented the source code of the application under test to detect runtime crashes.

III SPAG-C

Architecture Overview

As illustrated in Fig. 3.1, we have two sets of components: hardware components and software components.

Hardware Components

The DUT is the Android device that runs the application for which the test cases are written. It is worth noting that even though the test cases are written for a specific application, the DUT is what is being tested. The camera is used to capture the required GUI states during both record and replay phases. To avoid any interference with the process of capturing the required images, test cases are recorded by controlling the DUT remotely from a computer.

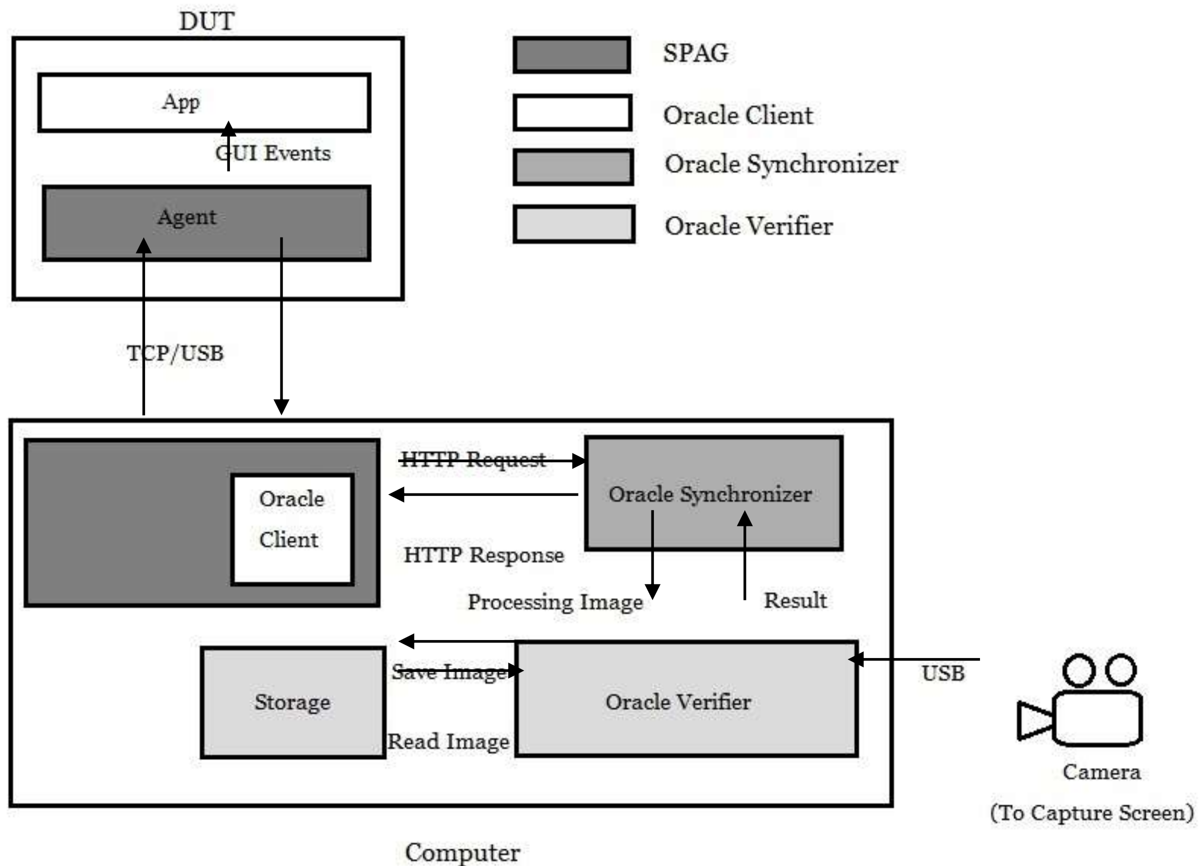
Software Components

SPAG-C records and replays test cases remotely. We divide the test oracle into three major components: oracle client, oracle synchronizer and oracle verifier. As shown in Fig. 3.1, the oracle client is coupled with the record-replay component, in this case SPAG, and it is in charge of automatically adding checkpoints to SPAG's test cases during the record phase and sending requests

to the oracle synchronizer to verify a GUI state during the replay phase. The oracle synchronizer uses Web service technology to expose the oracle verifier to the oracle client. Oracle synchronizer also handles requests from the oracle client, passes them to the oracle verifier, and sends the response back to the oracle client. The oracle verifier validates the testing results by capturing images from a camera, as shown in Fig. 3.1, and comparing the GUI states using the image comparison techniques previously discussed. Fig. 3.1 also demonstrates the original architecture of SPAG which consists of two modules: one runs on the DUT and the other on the host computer.

The agent, which is installed on the DUT, is in charge of capturing the required information to perform the verification process, while Sikuli IDE (integrated with Android Screencast) runs on the host computer, and is in charge of recording and replaying user events. Before moving forward to explain how SPAG-C's oracle works it is important to explain the differences between the way SPAG and SPAG-C verify the GUI states. Each of these two tools provides both semi-automatic and automatic verification methods. Both SPAG's and SPAG-C's semi-automatic verifications require testers to provide screenshots and write checkpoints into the test case. The difference is that SPAG captures these screenshots from the frame buffer, while SPAG-C does it from the camera.

The automatic method differs a lot in these tools. While SPAG simply performs a string comparison to verify that the same activity transition that occurred after the input of a specific event during recording also happens during replaying, SPAG-C automatically performs image comparison based on the impact the user events have on the GUI during recording. For example, if during the record phase the tester performs an event that causes the application to go from activity "com.android.contacts" to activity "com.android.- contacts.twelvekeydialer" then SPAG will corroborate that the same transition happens after replaying that event. This, however, does not ensure it is displayed correctly. SPAG-C, on the other hand, automatically takes the required screenshots based on the difference threshold set by the tester and performs image comparison, which provides a better evaluation of the GUI.



C-Fig. 3.1. Architecture Design of SPAG

SPAG-C Implementation

Oracle Client

As recounted before the oracle patron is coupled with the recordreplay element. In our case that factor is SPAG. Seeing that SPAG is implemented in Java, the oracle consumer can be applied in Java. Most of the code of the oracle purchaser is mechanically created via the software "wsdl2java" which is a part of the Apache CXF framework [22]. "wsdl2java" simply takes the WSDL file uncovered through an internet carrier (in this case the oracle synchronizer), and robotically generates Java code from which to name the provider. With that code in place, we introduced event listeners to SPAG to routinely name the oracle

synchronizer after any mouse movements in an effort to participate in the automated verification process described in the previous chapter. In the course of the replay phase, when the experiment case execution reaches a checkpoint, the consumer will make a name to the oracle synchronizer asking it to perform the requested action.

Oracle Verifier

The oracle verifier makes use of open source computer vision and .NET framework functionality to compare, crop, and rotate images. In particular, we use EmguCV, a cross-platform .NET wrapper for the OpenCV image processing library [24] that allows us to call OpenCV functions from any of the .NET compatible languages (in this case C#). OpenCV is a library of programming functions for real time computer vision. In this work we only use OpenCV's functions to extract and compare SURF features, to calculate and compare image Histograms, to perform template matching, and to detect Canny Edges.

In the previous chapter, we described most of the SURF comparison process; however, there are some implementation details that are worth mentioning. There are many ways to match SURF features. In this work we use OpenCV's BruteForceMatcher to perform KnnMatch (k-nearest neighbor match). KnnMatch returns the K nearest neighbors, K ¼ 2 in our case, based on euclidean Distance (L2 Distance). After the match is performed, we filter the matched features by using OpenCV's function VoteForUniqueness, which discards non-unique matches, with a uniqueness threshold of 0.9; and VoteForSizeAndOrientation, which discards those features whose size and orientation does not match the majority's size and orientation, using a scale increment of 1.5 and 20 rotation bins. In order to perform color histogram comparison, we need to first extract the color histogram of each image. However, we first reduce the colors of the images to get a reliable similarity measure and improve computation efficiency on each one of the three channels of the image: red, green, and blue.

After reducing the colors, we calculate the color histogram by calling OpenCV's DenseHistogram.Calculate method on each of the channels and compare the histograms by calling the cvCompareHist method. As described in Section 2, template matching finds a given small image in a larger image. However, since our automatic verification automatically decides what images will be used as the expected states the tester cannot provide the templates. Therefore, we automatically split the expected state into smaller images, and match each of those small images against the current state. This means that several template matches are performed; if at least one of the matches has a value smaller than the provided similarity threshold then the overall match is considered a failure.

Oracle Synchronizer

We use Microsoft WCF and C# to implement the oracle synchronizer. Microsoft WCF is a framework for building service-oriented applications that facilitates building interoperable Web services using different standards. The oracle synchronizer exposes three methods: Add- Checkpoint, VerifyState, and CaptureScreen. Each of the methods receives a different set of parameters. The Add-Checkpoint method is only called during the record phase, it receives the difference threshold as well as the previously captured screenshots cjp and cjp_1 , and performs the automatic verification process. The VerifyState method is only called during the replay phase, it receives the expected state sjp , captured during the record phase and the similarity threshold, and performs the verification process. Finally, the CaptureScreen method is called during both phases, it does not receive any parameters, but simply captures a screenshot and returns its path. Additionally, the oracle synchronizer is in charge of logging all relevant information each time there is a call to the service. For example, during the verification process, the oracle synchronizer will log the images that have been compared, the similarity thresholds we have, the length of time the image comparison process took, and the success or failture of the verification process.

Image Maintenance

SPAG-C captures a lot of images but not all of them are required in order to replay a test case. In order to automate image maintenance, we name images randomly by using. NET's Path.GetRandomFileName method, and we prefix image names with the word "record", in case of sjp , and "replay", in case of s0jp . Of all the images captured during the record phase, only the expected states are kept, and the rest are all discarded automatically. Expected states are required in order to replay a test case; therefore, these images are not deleted.

IV RESULTS

In the previous work SPAG we worked with SPAG with the help of Sikuli for GUI verification process and with Monkeyrunner. It may be hard to tell the results that are successful when using Testdroid, because of it is a commercial cloud service is not available publicly. GUITAR was also automatically generating test cases for a given UI. It basically adopts and uses object identification to perform verification, but it may not be useful for device testing.

As mentioned before, SPAG uses 3 hardware components: DUT, a Computer and an external camera. In order to make a fairer comparison with SPAG, we run our experiments on 2 different DUTs: MOTO Droid Turbo 2 and ASUS Zenfone 5. Where Droid Turbo 2 has a 5'4" in AMOLED capacitive touchscreen with 16M colors and a resolution of 1440X2560 pixels. Droid Turbo 2 runs on Android 7.0 with API level 25. On the other hand, ASUS Zenfone 5 has a 5.0 in IPS LCD capacitive touchscreen with 16M colors and resolution 720X1280 pixels and runs on Android 5.1. The external camera used is a 3.1MP Panasonic camera. We used a normal laptop as a host computer with 2.10GHz Inter Pentium B950 processor 2GB of RAM, Windows 10.

The test cases are created for Contacts, ShareIT, GooglePlay and Alarm Clock. During the record phase , we record every test case 10 times for each of the following different thresholds:20, 40, 60, 80 percent. During the replay phase, we replay the test cases for 100 times: 50 times to measure false positives and 50 times to measure false negatives.

The camera placed 12cm from the devices screen and the autorotation of the device was disabled and the brightness is fixed to 100 percent, so the images captured from the camera are in same resolution and in similar frames of the device.

In the testing, many no of checkpoints added to a test case in the record phase. Automatic verification always capture initial and final states of the application when the record process starts and finishes, respectively. As no of checkpoints added to the test case decreases as the different threshold increases.

V CONCLUSION

The test oracle is non-instructive, it does not depend on the DUT to perform the verification process. So it can be used to test a great variety of heterogeneous smart phones. We propose using Web services as an interface between both. So that the test oracles could be reused by other tools via Web services and more time can be saved from that. Reusing the test oracle is relatively simple because of Web service technologies. When we place the code, then we need to add event listeners to the record-replay tool to call the test oracle.

Sometimes the testing is difficult as the lighting conditions in the room changes like the reflection on the screen and the background of the device. In order to overcome that, we need to put it in a constant place, provide better condition.

So the experiments shows that the recording test case using SPAG-C's automatic verification is as fast as SPAG's but more accurate since we also make sure the application is being properly displayed. The testing is differ with the SPAG by 2 times or more, approximately. Using camera giving the results more accurate, yielding less than 2 percent false negatives/positives.

VI REFERENCES

[1] Microsoft MSDN. (2013, Mar.). Guidelines for touch interaction [Online]. Available: http://msdn.microsoft.com/en-us/library/cc872774.aspx

[2] A. M. Memon, M. E. Pollack, and M. L. Soffa, "Automated test oracles for GUIs," ACM SIGSOFT Softw. Eng. Notes, vol. 25, pp. 30–39, 2000.

[3] L. Baresi and M. Young, "Test oracles," University of oregon, Dept. of Computer and Information Science, Tech. Rep. CISTR-01-02," Eugene, OR, U.S.A., [Online]. Available:http://www.cs.uoregon.edu/michal/pubs/oracles.html, Aug. 2001.

[4] Q. Xie and A. M. Memon, "Model-based testing of communitydriven open-source GUI applications," in Proc. IEEE 22nd Int. Conf. Softw. Maintenance, Sep. 2006, pp. 145–154.

[5] T. Takala, M. Katara, and J. Harty, "Experiences of systemlevel model-based GUI testing of an android application," in Proc. IEEE 4th Int. Conf. Softw. Testing, Verification Validation, 2011, pp. 377–386.

[6] Y. D. Lin, T.-H. Chu Edward, S. C. Yu, and Y. C. Lai, "Improving the accuracy of automated GUI testing for embedded systems," IEEE Softw., vol. 31, no. 1, pp. 39–45, Jan. 2014.

[7] M. Grechanik, Q. Xie, and C. Fu, "Creating GUI testing tools using accessibility technologies," in Proc. IEEE Int. Conf. Softw. Testing, Verification, Validation Workshops, 2009, pp. 243–250.

[8] J. Kaasila, D. Ferreira, V. Kostakos, and T. Ojala, "Testdroid: Automated remote UI testing on android," in Proc. 11th Int. Conf. Mobile Ubiquitous Multimedia, Ulm, Germany, 2012, pp. 28:1–28:4.

[9] T.-H. Chang, T. Yeh, and R. C. Miller, "GUI testing using computer vision," in Proc. SIGCHI Conf. Human Factors Comput. Syst., Atlanta, GA, USA, 2010, pp. 1535–1544.

[10] Android screencast, an open-source remote control tool for Android devices. (2013, Mar.) [Online]. Available: http://code.google.com/p/androidscreencast/

[11] G. Pass and R. Zabih, "Comparing images using joint histograms," Multimedia Systems, vol. 7, no. 3, pp. 234–240, May 1999.

[12] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," Comput. Vis. Image Understanding, vol. 110, pp. 346–359, 2008.

[13] Opencv, template matching. (2013, Mar.) [Online]. Available:http://docs.opencv.org/doc/tutorials/imgproc/histograms/template_matching/template_matching.html

[14] T. Takala, M. Katara, and J. Harty, "Experiences of system-level model-based GUI testing of an android application," in Proc. IEEE 4th Int. Conf. Softw. Testing, Verification Validation, 2011, pp. 377–386.

[15] Android MonkeyRunner. (2013, Mar.) [Online]. Available: http://developer.android.com/tools/help/monkeyrunner_concepts.html

[16] Robotium framework homepage. (2013, Mar.) [Online]. Available:http://code.google.com/p/robo tium/

[17] Android GUITAR, a model-based system for automated GUI testing. (2013, Mar.) [Online]. Available: http://sourceforge.net/apps/mediawiki/guitar/index.php?title¼GUITAR_Home_Page

[18] Android developer guide, instrumentation. (2013, Jun.) [Online]. Available: http://developer.android.com/reference/android/app/Instrumentation.html

[19] Android developer guide, application fundamentals. (2013,Mar.) [Online]. Available: http://developer.android.com/guide/components/fundamentals.html

[20] A. M. Memon, "GUI testing: Pitfalls and process," IEEE Comput., vol. 35, no. 8, pp. 87–88, Aug. 2002.

[21] W3C working group note 11 February 2004, web services glossary[Online]. Availabe:http://www.w3.org/TR/ws-gloss/, Apr. 2013.

[22] Apache CXF: An open-source services framework. (2013, Apr.) [Online]. Available:http://cxf.apache. org/

[23] Windows communication foundation. (2013, Apr.) [Online].
Available: http://msdn.microsoft.com/en-us/library/ms731082.aspx
[24] EmguCV. (2013, Apr.) [Online]. Available:http://www.emgu.com/wiki/index.php/Main_Page
[25] R. Laganiere, OpenCV 2 Computer Vision Application Programming Cookbook. Birmingham, U.K.: Packt Publishing, May 2011.
[26] W3C working group, web services architecture. (2013, Jun.) [Online]. Available:http://www.w3.org/TR/ws-arch/#id2260892
[27] Android, hierarchy viewer. (2013, Jun.) [Online]. Available: http://developer.android.com/tools/help/hierarchy-viewer.html
[28] SPAG-C live demo. (2013, Jun.) [Online]. Available: http://youtu.be/V841LpD4ULo
[29] C. Hu and I. Neamtiu, "Automating GUI testing for android applications," in Proc. 6th Int. Workshop Autom. Softw. Test, 2011, pp. 77–83.
[30] S. Anand, M. Naik, M. J. Harrold, and H. Yang, "Automated concolic testing of smartphone apps," in Proc. ACM SIGSOFT 20th Int. Symp. Found. Softw. Eng., 2012, pp. 59:1–59:11.
[31] W. Choi, G. Necula, and K. Sen, "Guided GUI testing of android apps with minimal restart and approximate learning," in Proc. ACM SIGPLAN Int. Conf. Object Oriented Program. Syst. Lang. Appl., New York, NY, USA, 2013, pp. 623—640.
[32] N. Mirzaei, S. Malek, C. S. Pasareanu, N. Esfahani, and R. Mahmood, "Testing android apps through symbolic execution," ACM SIGSOFT Softw. Eng. Notes, vol. 37, pp. 1–5, 2012.
[33] R. Mahmood, N. Esfahani, T. Kacem, N. Mirzaei, S. Malek, and A. Stavrou, "A whitebox approach for automated security testing of android applications on the cloud," in Proc. 7th Int. Workshop Autom. Softw. Test, 2012, pp. 22–28.
[34] A. Machiry, R. Tahiliani, and M. Naik, "Dynodroid: An input generation system for android apps," in Proc. 9th Joint Meeting Found. Softw. Eng., Saint Petersburg, Russia, 2013, pp. 224 234.
[35] D. Amalfitano, A. R. Fasolino, and P. Tramontana, "A GUI crawling- based technique for android mobile application testing," in Proc. IEEE 4th Int. Conf. Softw. Testing, Verification Validation Workshop, 2011, pp. 252–261