IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 21, 2018, accepted July 3, 2018, date of publication July 10, 2018, date of current version July 30, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2854754

Robotic Arm-Based Face Recognition Software Test Automation

DEBDEEP BANERJEE, (Member, IEEE), AND KEVIN YU
Qualcomm Technologies, Inc., San Diego, CA 92121, USA
Corresponding author: Debdeep Banerjee (debdeepb@qti.qualcomm.com)

ABSTRACT Facial recognition is a feature that uses facial detection algorithms to detect a face and then invokes facial recognition algorithms to try to match the person's face. First, a person's face needs to be enrolled; during enrollment, the person's facial details are saved in a database—in this case, a mobile phone. The facial recognition algorithm uses this database to match the currently presented face with the faces saved in the database. The efficiency of a facial recognition algorithm depends upon the speed at which it can detect and recognize faces. The problem we address in this paper is that a reliable, automated method for testing facial recognition features in mobile phones. Because the multimedia capabilities of smartphones have expanded phenomenally, the need for thoroughly testing facial recognition algorithms has become crucial. The uses of facial recognition can range from facial authentication for unlocking phones to security for a number of other applications. To meet these needs, a reliable, automated test for validating the facial recognition algorithms must be developed. The challenge for the software test team was to automate the test cases, which involve tilting the phone at specific angles from the test subject. The permissible angular movements of the phone are determined by the algorithmic specifications of the facial recognition algorithm. We tested scenarios involving multiple faces, motion blur, panning the phone in front of the test subject faces at various speeds, and so on. We adopted a robotic arm to perform the facial recognition test cases and developed software to program the robotic arm to test the phone's facial recognition software for functionality, performance, and stability as well as adversarial tests. This paper discusses the computer vision use case for facial recognition and describes how we developed an automated facial recognition test suite using a robotic arm.

INDEX TERMS Software engineering, software test automation, computer vision, software testing, robotics, robotics, automation.

I. INTRODUCTION

This paper discusses the design, development, and implementation of a test automation strategy to verify facial recognition features in mobile phones.

The challenge in facial recognition test automation is to validate both major steps of the facial recognition algorithm. These steps include (1) determining whether the facial recognition algorithm is able to correctly identify that a face is present and (2) whether the face can be recognized correctly. We use logged messages to identify whether the faces are being recognized correctly.

Adopting the robotic arm when executing the facial recognition test cases has enabled us to accurately create and repeat tests that involve tilting the tested device to specific angular positions or panning the tested device while the algorithm is running to detect faces while focusing on different test subjects involving multiple faces, etc.

Based on these goals for designing and developing the facial recognition algorithm we focused on the following areas.

A. ROBOTIC ARM-BASED TEST AUTOMATION FOR FACE OCCLUSION TEST

For the facial occlusion test, the facial recognition software should correctly recognize the face with the left eye, right eye or mouth region covered. The robotic arm precisely positions the device under test in front of the test subject. A rectangular patch is used to occlude specific portions of the face. The face recognition algorithm extracts the features of the partially occluded face and compares them with the

features saved in the face database. The face database stores all the facial features of the faces when they are enrolled. This test case is automated with the help of the robotic arm, and this test helps in testing adversarial scenarios such as facial occlusion.

B. TEST AUTOMATION USING A 6-JOINT PROGRAMMABLE ROBOTIC ARM FOR TESTING FACE RECOGNITION ALGORITHMS WITH ANGULAR MOVEMENTS (INCLUDING YAW, PITCH AND ROLL) WITH HIGH PRECISION

The robotic arm serves to automate the tedious work of manually testing facial recognition algorithms with angular movements of device under tests. Moreover, manual testing is error prone. Therefore, being able to programmatically control the 6 robot arm motors to achieve a specific angular movement has increased test efficiency and validity.

C. TEST AUTOMATION FOR VALIDATING FACE RECOGNITION ALGORITHMS ON LARGE DATASETS AND WITH HARDWARE ACCELERATION

We tested the face recognition algorithms with large datasets containing nearly twelve thousand faces with various angular movement patterns to assess the precision of the face recognition algorithms. We discuss these test results in the Results section of this paper. The face recognition algorithms can be tested with hardware-accelerated solutions.

The test automation also includes on-device test automation to control the phone's facial recognition application, which involves sending Google Android instrumentation-based commands to the phone to launch, close and applying various settings in the phone's software application.

We evaluated the software integration points and ensured that specific test cases are included at these software integration points. Adding test automation earlier in the software development cycle has helped us to identify software faults earlier in the development life cycle.

II. MOTIVATION

The goal in designing and developing the facial recognition test automation using the robotic arm is to ensure that we have a test framework suitable for accuracy, functionality, stability, and performance tests.

A. TEST AUTOMATION VALIDATION: PERFORMING FACE RECOGNITION TESTS AT VARIOUS TEST SUBJECT FACE ANGLES

We programmed the robotic arm so that it can pick up the test device and then perform angular motions around the faces of test subjects. This ability effectively allows viewing the test subject from various angles and supports determination of the ability of the software algorithm to detect these faces at various angles. We followed the software design document specifications with respect to the exact angles at which the algorithm should be able to detect and recognize target faces. Moreover, we programmed the robotic arm to test positive (within spec) and adversarial (beyond spec) angular movements.

B. EXECUTION OF STABILITY TEST CASES WITH ANGULAR MOVEMENTS FOR FACE RECOGNITION

Testing facial recognition algorithms manually with multiple test subjects at various angular movements over multiple iterations is both tedious and error prone. Substituting robotic arm-based test automation for facial recognition is highly effective in executing the stability test cases and discovering software issues.

C. EXECUTION OF THE FACE RECOGNITION TESTS BY INSERTING OF MOTION OR JITTER TO SIMULATE REAL WORLD SCENARIOS OF CUSTOMERS USING THE SOFTWARE

The robotic arm was programmed to insert motion and jitter while the device is pointed toward the faces of the test subjects. This process helps to simulate real-world testing scenarios in which the customer using the application on the phone may shake the device. We carefully quantify the issues revealed by these tests and then triage the failures as legitimate failures.

D. DESIGN AND DEVELOPMENT OF END TO END TEST AUTOMATION ON A DEVICE TO TEST FACE RECOGNITION ALGORITHMS

The face recognition test automation can launch the facial recognition application on the phone and it provides an interface to the on-device software to programmatically apply various application settings. The test automation postprocesses the logs generated from the tests to determine the results of each test case.

E. MANUAL TESTING OF FACE RECOGNITION ALGORITHMS MAY BE ERROR PRONE

Manual testing may be error prone and tedious. Since we need to test at various yaw, pitch and roll angles for face recognition performing these tests manually may lead to errors. The programmatic robotic arm has provided us an interface to develop test automation to test these angular movements.

F. PRECISION MAY BE LOST WITH MANUAL TESTING

An important point of testing image processing software is the precision calculation for performance tests. So, we must precisely position the mobile phone under test in front of the test subjects. The robotic arm gives us this feature.

G. MANUAL TESTING MAY HAVE ISSUES WITH TEST SCALABILITY

Executing multiple software products may be an issue with manual testing as it may be time consuming. Using a robotic arm can help us scale up test execution over several hours.

III. BACKGROUND AND RELATED WORK

Robots have long been employed by industry to automate repetitive tasks [1]. Software testing has also greatly benefited end-users through GUI (graphical user interface)-based testing using robots.

Using a robotic arm for software testing [2] is a prevalent approach, as the robot can be made to repeat a specific test sequence precisely [3]. Black-box testing involves testing a software program without knowing its internal implementation or its architecture. Software can be tested using black-box techniques when the inputs and the expected output are known. A robot setup can be efficiently used for software testing because it can precisely repeat the test sequences and the test results can then be analyzed [4]–[9].

Robots can be used as flexible, reconfigurable model. The robot achieves rectilinear locomotion by coupling structural deformation and directional friction, promoting a locomotion strategy that is ideal for traversing narrow channels [11], [12]. However, testing software using robot-based systems is challenging because it is necessary to test many of the characteristics of physical systems [13], [14].

The Robot Framework is open source software based on the Python language and comprises a keyword-driven automated test framework [14]. The challenges of robotic software testing extend beyond conventional software testing. Valid, realistic and interesting tests must be generated for multiple programs and hardware running concurrently and that are intended for deployment into dynamic environments [15].

Manual testing is a time-consuming process. In addition, regression testing, because of its repetitive nature, is error-prone; therefore, automation is highly desirable. Robot Framework is a simple but powerful and easily extensible tool that utilizes a keyword-driven testing approach [16].

A robot-based remote testing platform for mobile applications is presented in [17] that allows a tester to remotely interact with a mobile device by controlling a robot. Profiling tests must be executed for software, including executing conformance tests [18].

As an example, [19] presents an approach for behavioral specification mining based on sets of predetermined patterns observed over the collected data. Summarizing user sessions into probabilistic user models and their use for test generation is discussed in [20].

With respect to Android test automation, the automation efforts that have been developed such as [21]–[23]. Automated testing efforts have been applied to symbolic execution [24]–[26] to generate inputs for Android apps. Model-based testing has been widely studied in testing GUI-based programs [27]–[30].

In general, model-based testing requires users to provide a model of the app's GUI [31], [32], although automated GUI model inference tools tailored to specific GUI frameworks exist as well [33], [34].

For mobile applications, the analysis types can be classified as static and dynamic; there is growing interest in performing dynamic analyses of mobile apps (e.g., [35]–[37].

The approach to Android-based application testing enables test engineers to effectively automate the testing process and achieve comprehensive test coverage.

IV. SOLUTION

A. OVERVIEW

The face recognition software under test in this study has a standard algorithmic flow. It first takes a snapshot of the human face and then performs face detection to determine the location of the face in the acquired image. Then, it creates face patches and extracts features from these patches. Finally, depending on whether the task being tested is an enrollment task or a recognition task, the software either saves the features to the face database or matches the features against the existing database entries to display the best-matched face ID. Fig. 1 depicts the software's algorithm workflow.

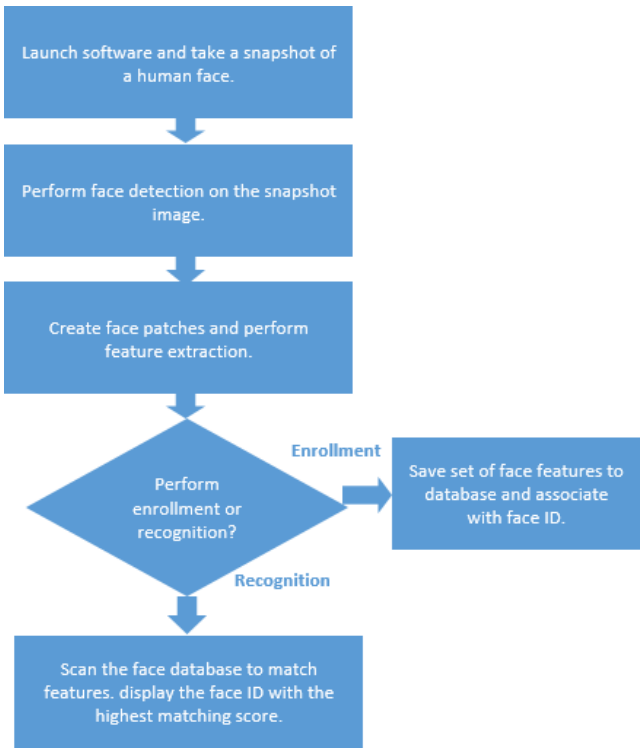

FIGURE 1. Workflow of the face recognition algorithm.

Our facial recognition test automation is designed to objectively determine whether faces are correctly recognized by the algorithm. The test automation process involves both on-device test automation development to control the application running on the phone's software and robotic arm programming.

B. OBJECTIVE

According to the facial recognition software documentation, the specified limitations for recognizing faces are as follows: pitch angles from −15° to 15°, roll angles from −45° to 45°, and yaw angles from −20° to 20°, as shown in Fig. 2.

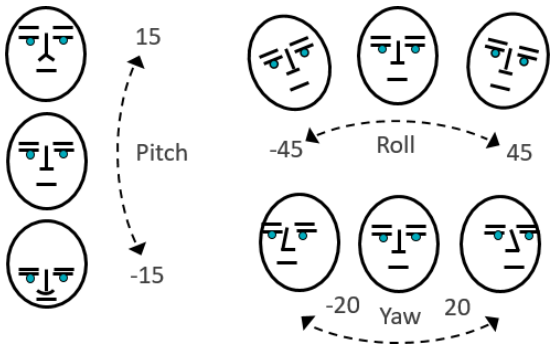

FIGURE 2. Supported angular ranges for the facial recognition software.

To properly test this software, we need to verify the facial recognition results within and at the supported angular limits. However, it is challenging for a human tester to manually move the test device to these angles without using proper measuring equipment. Thus, manual testing usually lacks precision and is very time consuming. Therefore, we developed automated tests using a robotic arm, which is able to move the test device precisely to the specified angles and test the facial recognition feature automatically.

C. TEST SETUP

We have a dedicated lab for computer vision use-case automation equipped with a DENSO VS-Series six-axis articulated robot. Using this robotic arm in our test automation has some clear advantages: it features high precision, which is important because we require it to be able to rotate a device to a certain angle. The robotic arm can be programmed to maintain the device at a precise distance from the objects for some test cases, and its compact design saves lab space. The six-axes-of-motion design makes the robotic arm highly flexible; it can perform continuous motions and angular motions that a human arm would find difficult. On test bench next to the robotic arm, are mannequins that we use as facial recognition test subjects. The mannequins are placed at fixed positions relative to the robotic arm: we can use these positions to program the robotic arm (see Fig. 3). On the opposite side of the test bench, there is a seat for testing facial recognition with a real person. In these live scenarios, the robotic arm moves the test device to the same face angles as when using a mannequin. We also implemented controllable lighting conditions and safety features for operating the robotic arm in this feasible testing environment.

D. AUTOMATION WORKFLOW

The design methodology for the high-level automation flow and the software we use is shown in figure 4.

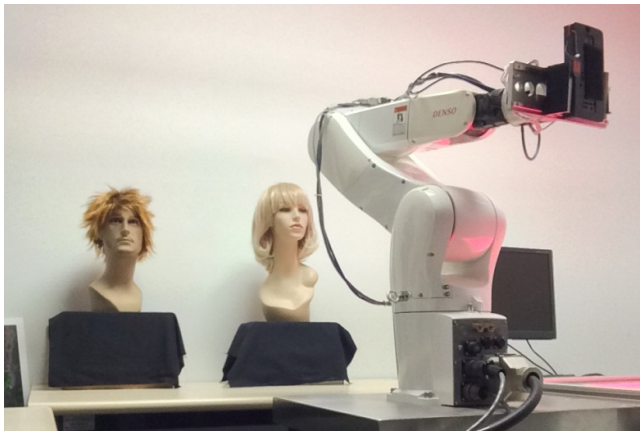

FIGURE 3. Robotic lab test setup.

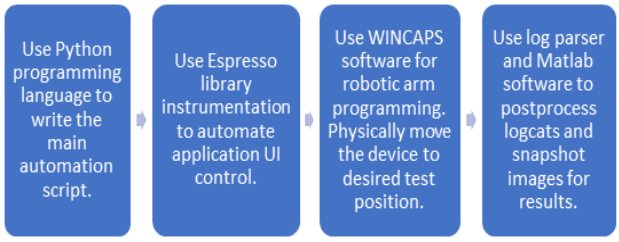

FIGURE 4. Design methodology for face recognition test automation.

To automate the complete test process, the software control sequence for the facial recognition app must also be automated. We use the Espresso library to instrument this software because it provides the ability to simulate user clicks, scrolling, selection and other actions through commands. These simulates actions are implemented by our main automation script.

When the main automation script starts, the PC first sends a signal to the robot controller to execute the robot problem that picks up the test device. Then, it begins to execute the main robotic program that instructs the robot to move the device to precise pitch/roll/yaw angles depending on the test case requirements. After the device is in the assigned position, the automation script triggers the facial recognition software. The app will launch, focus on the face detected in the preview image, begin the recognition process and display the recognized name on the preview image. The instrumentation then takes a snapshot of the preview and exits the app. All these processes are logged through logcat and saved to the PC running the automation test. This automated test can repeat multiple times while moving the device from the positive to the negative angular limit. For example, in the pitch angle test, we repeat these steps at 15°, 10°, 5°, 0°, −5°, −10° and −15° angles, which ensures that the different angles within the limits are covered and thoroughly tested by the test automation script. After the robot arm has moved the device through all the required angles, it releases the test device at the point where it was picked up and moves back

to a resting position. The automation script saves the result snapshots and purges the logcat for postprocessing. At the end of the automation process, the script returns a result of ''pass'' or ''fail'' depending on whether the facial recognition software was able to correctly recognize the mannequin, or a real person enrolled prior to the test. The automation flowchart is depicted in Fig. 5.

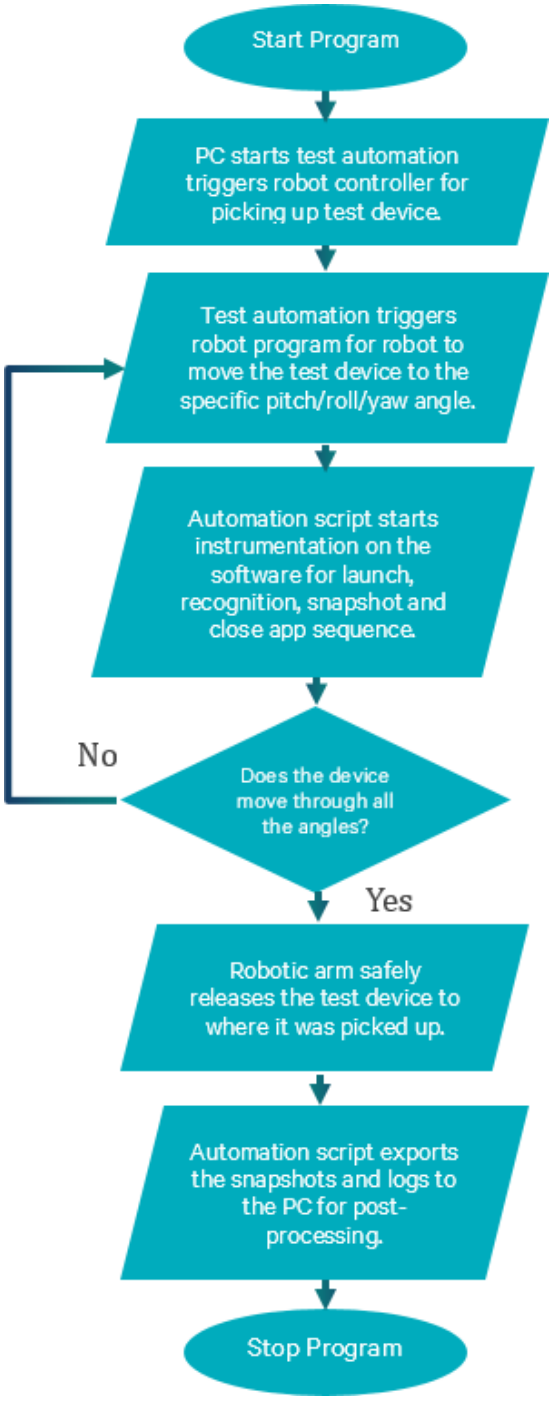

FIGURE 5. Robotic arm test automation script workflow for facial recognition.

E. PROGRAMMING THE ROBOTIC ARM

The robotic arm plays a major role in this test automation scenario. We programmed the robotic arm using WINCAPS 3 software, which makes programming the robot easier through its built-in 3D arm view, where it simulates the robotic arm position and movements and displays the corresponding coordinates. Thus, a developer can know where the robotic arm is and how the program will move the arm. It is also very helpful to create 3D models in this simulated environment to provide a good simulation of both the robotic arm and the test subject. Because we are programming the robot arm to scan a mannequin, we create a 3D model that has a similar shape and size. The model's position is measured using the same scale as the simulation coordinates. Consequently, we can use simple math and trigonometry to calculate the coordinates required by the robot arm.

When programming a pitch angle of 15°, the tip of the robotic arm that holds the device must be raised 15° above the level of the mannequin's eyes. In this test program, we want to maintain a constant distance of 50cm between the device and the mannequin. Given the known distance between the robotic base and the mannequin (75 cm) and the distance between eye level height and the robot arm base (60 cm) we need to calculate the desired x-axis and z-axis coordinates of the robotic arm's tip. Because the mannequin is aligned directly in front of the robotic arm, the y-axis coordinate is set to 0. Please refer to Fig. 6 for the calculation diagram.

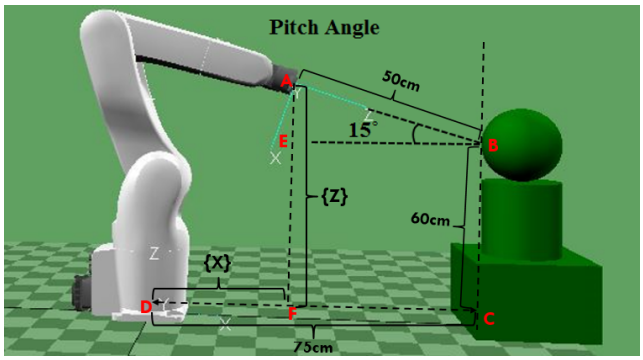

FIGURE 6. Simulation of robotic arm at a 15° pitch angle above eye level.

To calculate the x-axis coordinate (line DF in Fig. 5) because the length of CD is known, we need to find the length of CF which is equal to the length of BE. Since triangle ABE is a right triangle and the length of its hypotenuse is known (50 cm), one of its angles is known (15°), we can use trigonometry to calculate the length of BE:

$$BE = AB \times \cos 15^\circ \approx 448.3\ cm,$$

After obtaining the length of BE, we know the coordinates along the x-axis (DF):

$$DF = CD - CF = CD - BE = 75 - 48.3 = 26.7\ cm$$

Our next goal is to calculate the z-axis coordinate (AF). Using the same approach, the length of AE (the other side of the

right triangle ABE) can be calculated, thus also allowing the determination of the length of AF:

$$AE = AB \times sin15^\circ \approx 12.9\ cm$$
$$AF = AE + EF = AE + BC = 12.9 + 60 = 72.9\ cm$$

Given the x, y, and z coordinates of the tip of the robotic arm {26.7, 0, 72.9}, the simulation software can calculate the rotational angles of each of the 6 robot joints required to move the arm's tip to the specified coordinates.

Using the same approach, we can program the robotic arm to move to positions at pitch angles of 10°, 5°, 0°, −5°, −10° and −15°. For yaw angles, rather than moving up and down, the robot needs to move from left to right in an arc path at the mannequin's eye level, as shown in Fig. 6. For roll angles, the robotic arm needs to rotate its last joint to the desired angle while holding the device parallel to the mannequin's face, as shown in Fig. 7.

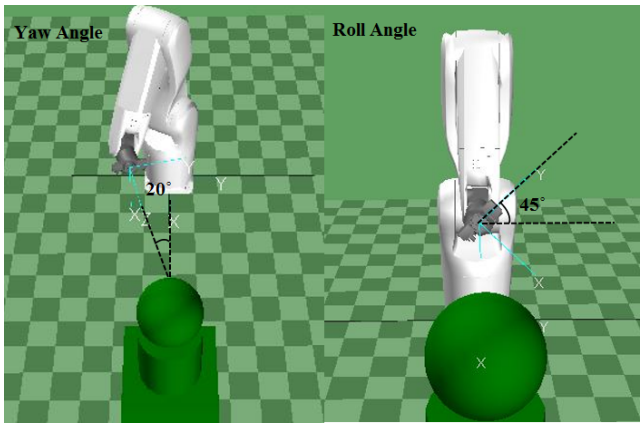

FIGURE 7. Robotic arm at a 20° yaw angle and a 45° roll angle.

After setting all the angular positions, we then program the robotic arm to move between each angle. A pause interval is added at each stopping point to allow time for the face recognition software to launch the app, execute the recognition test and close the app. This approach ensures that the software exercises a fresh recognition at every angle instead of simply tracking a recognized face from other angles. Fig. 8 demonstrates how the program controls the robotic arm to move the test device to the pitch, yaw and roll angles on mannequin No. 1.

In our case, we want to increase test coverage by placing a second, male mannequin on the setup bench, as shown in Fig. 9. The same calculation method can be used to position the robotic arm for the second mannequin—we just need to add the coordinates of this mannequin to the reference positions during the calculation. This addition adds some complexity to the math; however, it also allows a substantial expansion of the test models when space allows while still using a single robotic arm. Using two mannequins eliminates the effort needed to manually switch test models, and the entire automated process can be triggered remotely. This robotic arm can also turn 180° and repeat the same set

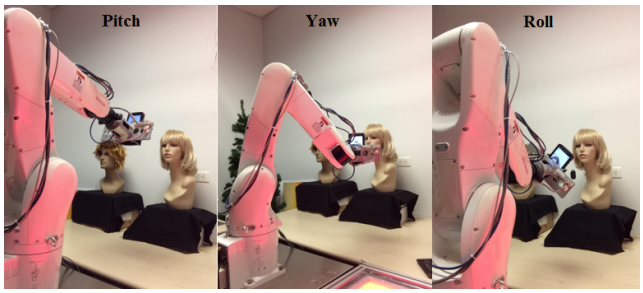

FIGURE 8. Robotic arm at pitch, yaw, roll angles on mannequin No. 1.

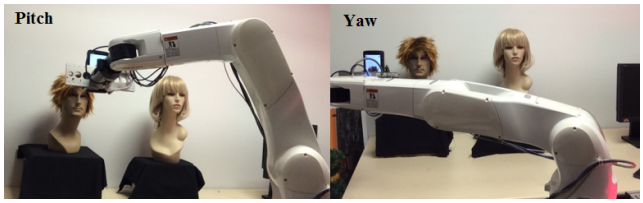

FIGURE 9. Robotic arm at pitch and yaw angles on mannequin No. 2.

of angles. In addition, a real person can sit at a fixed position and launch a live recognition, thereby making the automation setup even more practical.

F. TEST AUTOMATION ON FACE OCCLUSION USE CASE

Another key use case for applying robotic arm automation is when testing facial occlusion. Facial occlusion scenarios have always been a challenging test for facial detection and facial recognition algorithms. In many real-life scenarios, not all parts of the human face can be clearly seen. Sometimes, the eyes and ears are partially covered by hair, the mouth and nose may be covered by a pollution mask, and hands may also interfere and block part of the face when performing facial recognition on images, videos or live subjects. Thus, occlusion forms a challenge to facial recognition algorithms' precision and performance.

Facial feature extraction is one of the software algorithm processes previously mentioned. During feature extraction, the algorithm locates the facial regions unique to each individual, called facial patches, as shown in Fig. 10. The extracted patches are either stored or compared to the facial database during facial matching.

In facial recognition testing, we want to block each of these regions of the face in turn to simulate facial occlusion scenarios that occur in real-life to evaluate how the facial recognition software performs. With 2-dimensional images, it is easy to block any region of the face with image editing tools and therefore automate the process. However, blocking these facial regions with a 3-dimensional face model is more challenging without manually covering the face. Using the robotic arm makes it possible to automate this process.

In our lab, we use a mannequin in a fixed position as the facial model and use a rectangular light-blocking material as cover that is also fixed in position relative to the

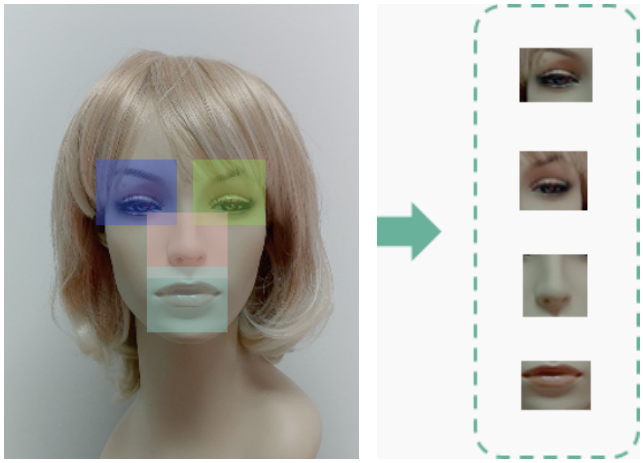

FIGURE 10. Diagram of the facial patches of a mannequin.

mannequin's face. By positioning the test device with the camera at specific angles with respect to the face, the covering will block an exact region of the facial image with respect to the camera. As shown in Fig. 11, position P1 will cover the left eye region, P2 will cover the right eye region, P3 will cover the nose region, and P4 will cover the mouth region.

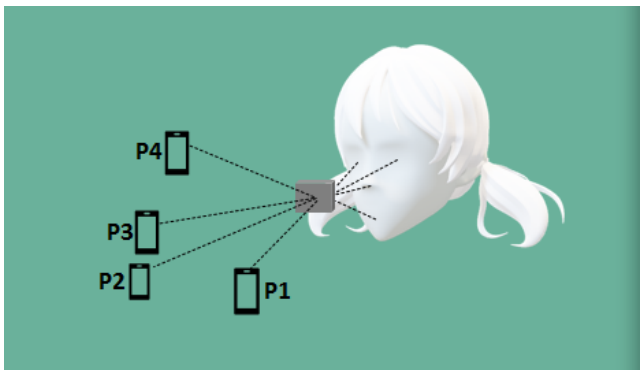

FIGURE 11. Diagram of the four camera positions for testing facial occlusion with a single cover.

Since we only need to move the test device to achieve the desired facial occlusion scenarios, this test process can be automated by programming the robotic arm to move the device to positions P1, P2, P3 and P4. Depending on the mannequin's position and the cover's position, height and distance, the angle and position required for the device can be calculated or estimated visually by checking the preview image. The angle formed between the camera and the center of the cover will be the same angle as that from the center of the cover to the covered facial region. The size of the covered region depends on the size of the covering material and the distance ratio of the cover position between the mannequin and the camera. After determining the 4 positions of the camera that yield the desired facial occlusions, we store them into the robotic arm's script and move the robotic arm

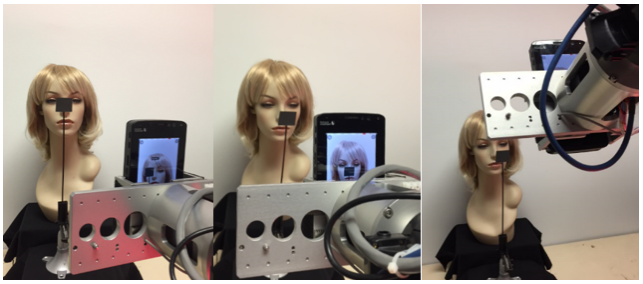

FIGURE 12. Robotic arm moves the device to each position for face occlusion testing.

through each of these positions to perform the automated facial occlusion test illustrated in Fig. 12.

G. TEST AUTOMATION ON 2D IMAGES

Although face recognition automated software testing garners major benefits from using robotic arm in angular and occlusion use-cases on 3D faces, there is a limit to the number of 3D test subjects we can test to validate the performance of the facial recognition software. Therefore, we also designed robotic arm-based automation testing on 2D face images by focusing the device on a monitor screen, as shown in Fig. 13.

FIGURE 13. 2D face image testing on display monitor.

The advantage of using a monitor screen is that it allows facial recognition testing on a large face image dataset. The FERET dataset [38], [39] is one of the commonly used datasets for evaluating facial recognition software. It contains images of 1,200 individuals for face enrollment, with a total of 12,827 face images from these individuals that can be used to test facial recognition. The FERET recognition images were divided into 5 groups based on face yaw-angle ranges. We display the enrollment images on the monitor as a slide show and then use the end-to-end face recognition software to enroll the faces in its database. Subsequently, we display the recognition images to test the software's recognition performance. The results are parsed by the automation script for analysis. Using this automation setup, we can evaluate and compare devices with different face recognition solutions.

Face image manipulation testing is another use case that can benefit from this robotic arm and monitor setup. For example, we can easily blur the face images or zoom-in and zoom-out on the face images, and then run the test automation suite on the altered images to evaluate the facial recognition software's performance on these types of scenarios, as shown in Fig. 14.

FIGURE 14. Face image manipulation scenarios.

The brightness of the monitor display can also be changed to introduce another factor that can alter the performance results. Our test setup allows multiple test option possibilities using both 2D face images and 3D face models. These testing options will benefit the coverage and quality of facial recognition software.

V. RESULTS

Using the automation setup with the robotic arm, we can easily perform functional and performance tests using the facial recognition software in different scenarios. We collect the results in term of true positives, true negatives, false positives and false negatives. The facial recognition time limit is set to 300 milliseconds. If the software cannot display the correct name associated with a face within that time limit, it will generally be treated as a false negative.

A. 3D FACE MODELS WITH ANGLES

Facial recognition tests at various angles are functional tests; for these, it is best to use the robotic arm with the 3D mannequin setup. We ran the test automation at 15° to −15° pitch angles, 45° to −45° roll angles and 20° to −20° yaw angles. At each angle, we executed the recognition process 100 times to determine how many times the software correctly identified the mannequin. After the test script completed, we reviewed the collected test results, which consist of the test logs and the screenshots as shown in Fig. 15. The screenshots help us confirm that the application is displaying the recognized names correctly at the end to end level. At every angle, the face remains correctly positioned in the preview image, which means that the robotic arm positions were calculated and executed precisely as intended.

We preserved only the true positive rates because these were functional test cases, and we enrolled only two mannequins to test whether the software could recognize them at the specific angles. From the test results shown in Table 1, the face recognition succeeded every time within ±10° pitch

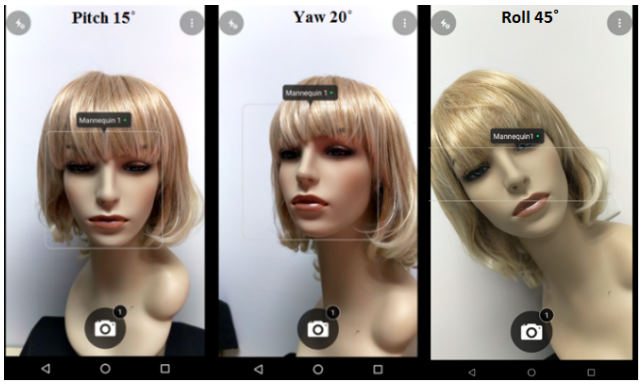

FIGURE 15. Test result screenshots at different angles from mannequin 1 & 2.

TABLE 1. Performance test results at the various pitch, yaw, and roll angles.

Pitch Angles	True Positive	Roll Angles	True Positive	Yaw Angles	True Positive
15°	99%	45°	98%	20°	95%
10°	100%	30°	99%	15°	97%
5°	100%	15°	100%	10°	99%
-5°	100%	-15°	100%	-10°	99%
-10°	100%	-30°	100%	-15°	98%
-15°	99%	-45°	98%	-20°	96%

angles and failed only twice at ±15° pitch angles. the roll angles had a true positive rate of 98% at ±45°. A slight performance impact occurred as the yaw angles approached the specification limits: the recognition success rate was 95% for 20° yaw angles, and 96% for −20° yaw angles. The yaw angles proved to be more challenging for facial recognition compared to the pitch and roll angles. Thus, the software should undergo further testing for facial recognition performance evaluation. However, this issue can be overcome by enrolling faces from both frontal and side views.

Using this robotic arm automation setup, we can expand the test cases to recognize faces at angles beyond the supported limits. Being able to precisely push these limits will help software developers test solutions that may improve facial recognition performance and, therefore, enhance their algorithms.

B. 3D FACE MODELS WITH OCCLUSIONS

On the facial occlusion test, the facial recognition software is able to correctly recognize the mannequin's face with the left eye, right eye and mouth region covered. However, we discovered that when the nose region is covered, the software can no longer detect the face, as illustrated in Fig. 16. This may be because the nose region is essential for the facial detection algorithm. Additionally, because the nose region is

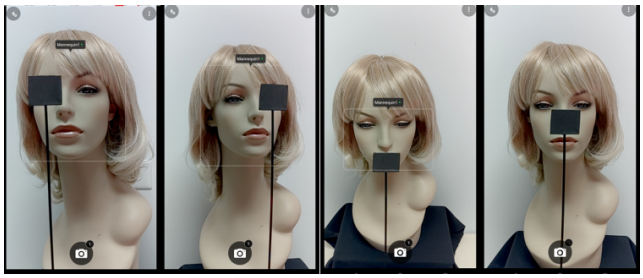

FIGURE 16. Test result screenshots of facial occlusion scenarios on a mannequin.

the highest point of human face geography, it is the optimal viewing position for human facial recognition.

From these results, we can see that the robotic arm is a reliable and effective solution for fully automated facial occlusion testing. It is also possible to expand the use cases by altering the size of the facial covering material to assess the software's limitations with regard to facial occlusions.

C. TESTING 2D FACE IMAGES IN THE FERET DATASET

Large 2D face image datasets such as FERET are ideal for performing facial recognition performance evaluations. Using the robotic arm and the display monitor setup, we executed automated tests on the 5 categories of face data sets based on face yaw angles. There were 4,010 face images within the 0–20-degree yaw angles, 1,926 images within 20–30-degrees, 1,013 images within 40–50 degrees, 2,935 images within 60–70 degrees and 2943 images within 80–90-degree yaw angles. The test results are shown in Table 2.

TABLE 2. Face recognition performance result on the FERET dataset.

FERET Database Face Yaw Angles	Face Detection Rate	True Positive	True Negative	False Positive	False Negative
0-20 Degree	99.18%	94.74%	1.63%	1.73%	1.89%
20-30 Degree	90.81%	69.07%	11.72%	0.80%	18.41%
40-50 Degree	68.41%	39.83%	19.05%	0.14%	40.98%
60-70 Degree	41.26%	11.31%	12.55%	0.00%	74.82%
80-90 Degree	1.90%	0.00%	1.79%	0.00%	98.21%

As Table 2 shows, the face detection success rate reaches 99% within the 0–20-degree yaw angles with an approximately 95% true positive recognition rate. The false positive rate is at 1.7%. These angles are well within the supported yaw angle range of the face recognition software we tested, and the performance is good at this range. The other 4 yaw-angle categories are outside of the supported range; thus, the performance can be expected to decrease, as shown in Fig. 17.

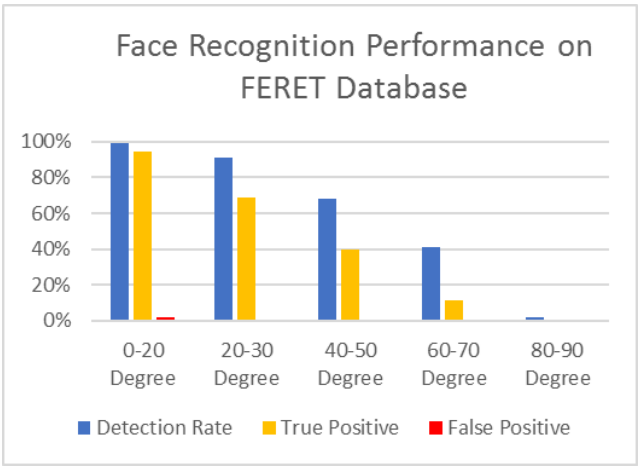

FIGURE 17. Graph of the face recognition software performance results.

This graph shows the face detection, true positive and false positive rates; these 3 are the key performance metrics for face recognition. At 20–30-degree face yaw angles, the face detection rate is still good, reaching approximately 91%; however, the true positive rate drops to 69%, almost a 25% decrease compared to the previous category. This is expected, because the more a face turns, the fewer facial features it reveals, which makes it harder for a facial recognition algorithm to recognize the face. From the graph in Fig. 16, we can see a consistent rate of decrease in both the face detection rate and the true positive recognition rate throughout the first 4 categories. At 80–90-degree yaw angles, only one side of the face is visible; at this point face detection and face recognition are expected to fail. The false positive rate remains low for all the test images, which is good, because the facial recognition software should not falsely accept an incorrect face—especially when the facial recognition software is used for security purposes. Overall, we obtained meaningful and reliable results from test automation.

VI. CONCLUSION

Facial recognition algorithms require a comprehensive testing strategy to evaluate the effects of algorithmic code changes at both the application programming interface (API) level and at the end-to-end application level. The developed facial recognition test automation using the robotic arm provides us with a testing framework that can efficiently test churning software code to uncover software issues.

The facial recognition algorithm is tested using functionality, stability, performance and adversarial tests. The test automation involves on-device testing of the algorithms and provides test results based on log postprocessing as well as other characteristics. The usage of the robotic arm is a method for testing the face recognition algorithms and it can be implemented with other manipulating platforms also. The inclusion of the robotic arm has helped us to scale the test

automation process and execute tests across multiple software products.

VII. ADVANTAGES OF USING FACE RECOGNITION TEST AUTOMATION

In this study, we used a programmable robotic arm setup to perform panning movements in front of test subjects at different angles and speeds. The arm allows us to precisely control yaw, pitch and roll movements of the device under test in front of the test subjects, and the test scripts include postprocessing support to extract the names associated with recognized faces. Comparisons with the ground truth of the face detected in the image helps us to evaluate the test results. The test automation enables us to test real-life facial recognition scenarios (such as the inclusion of automated tests involving hand jitter motions) using the robotic arm. This capability helps us to effectively catch issues that can customers might experience.

The ability to pick up the phone under test and focus the phone on several test subjects to involve multiple faces in a session helps in testing prolonged sessions with multiple test vectors. The test automation is also used for stability testing by programming the robotic arm to perform multiple iterations holding the device under test in front of the test subjects' faces at various angles. This capability has helped us catch multiple race condition, memory corruption, and memory leak issues that occur only after several testing iterations.

Overall, the inclusion and deployment of the facial recognition test automation has increased our test coverage and helped us to catch software regressions.

REFERENCES

[1] K. Mao, M. Harman, and Y. Jia, "Robotic testing of mobile apps for truly black-box automation," *IEEE Softw.*, vol. 34, no. 2, pp. 11–16, Mar./Apr. 2017.

[2] D. Banerjee, K. Yu, and G. Aggarwal, "Hand jitter reduction algorithm software test automation using robotic arm," *IEEE Access*, vol. 6, pp. 23582–23590, 2018, doi: 10.1109/ACCESS.2018.2829466.

[3] D. Banerjee, K. Yu, and G. Aggarwal, "Robotic arm based 3D reconstruction test automation," *IEEE Access*, vol. 6, pp. 7206–7213, 2018.

[4] V. Garousi and M. Felderer, "Worlds apart: Industrial and academic focus areas in software testing," *IEEE Softw.*, vol. 34, no. 5, pp. 38–45, Sep. 2017.

[5] M. Harman, "Search based software testing for Android," in *Proc. IEEE/ACM 10th Int. Workshop Search-Based Softw. Test. (SBST)*, May 2017, p. 2.

[6] J. Guiochet, M. Machin, and H. Waeselynck, "Safety-critical advanced robots: A survey," *Robot. Auton. Syst.*, vol. 94, pp. 43–52, Aug. 2017.

[7] C. Yu and J. Xi, "Simultaneous and on-line calibration of a robot-based inspecting system," *Robot. Comput.-Integr. Manuf.*, vol. 49, pp. 349–360, Feb. 2018.

[8] M. Jasiński, J. Mączak, P. Szulim, and S. Radkowski, "Autonomous agricultural robot—Testing of the vision system for plants/weed classification," in *Automation*, vol. 743, R. Szewczyk, C. Zieliński, and M. Kaliczyńska, Eds. Cham, Switzerland: Springer, 2018.

[9] J. Brookes *et al.*, "Robots testing robots: ALAN-Arm, a humanoid arm for the testing of robotic rehabilitation systems," in *Proc. Int. Conf. Rehabil. Robot. (ICORR)*, Jul. 2017, pp. 676–681.

[10] N. Cramer *et al.*, "Design and testing of FERVOR: FlexiblE and reconfigurable voxel-based robot," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 2730–2735.

[11] J.-H. Lim, S.-H. Song, J.-R. Son, T.-Y. Kuc, H.-S. Park, and H.-S. Kim, "An automated test method for robot platform and its components," *Int. J. Softw. Eng. Appl.*, vol. 4, no. 3, pp. 9–18, Jul. 2010.

[12] M. Mossige, A. Gotlieb, and H. Meling, "Testing robotized paint system using constraint programming: An industrial case study," in *Proc. 26th IFIP Int. Conf. Test. Softw. Syst.*, 2014, pp. 145–160.

[13] M. Mossige, A. Gotlieb, and H. Meling, "Test generation for robotized paint systems using constraint programming in a continuous integration environment," in *Proc. IEEE 6th Int. Conf. Softw. Test., Verification Validation*, Mar. 2013, pp. 489–490.

[14] L. Jian-Ping, L. Juan-Juan, and W. Dong-Long, "Application analysis of automated testing framework based on robot," in *Proc. 3rd Int. Conf. Netw. Distrib. Comput.*, Oct. 2012, pp. 194–197.

[15] D. Araiza-Illan, A. G. Pipe, and K. Eder, "Intelligent agent-based stimulation for testing robotic software in human-robot interactions," in *Proc. 3rd Workshop Model-Driven Robot Softw. Eng.*, Leipzig, Germany, Jul. 2016, pp. 9–16.

[16] S. Stresnjak and Z. Hocenski, "Usage of robot framework in automation of functional test regression," in *Proc. 6th Int. Conf. Softw. Eng. Adv. (ICSEA)*, Barcelona, Spain, Oct. 2011, pp. 30–34.

[17] K. B. Dhanapal, "An innovative system for remote and automated testing of mobile phone applications," in *Proc. Service Res. Innov. Inst. Global Conf.*, 2012, pp. 44–54.

[18] S. Elbaum and M. Diep, "Profiling deployed software: Assessing strategies and testing opportunities," *IEEE Trans. Softw. Eng.*, vol. 31, no. 4, pp. 312–327, Apr. 2005.

[19] G. Reger, H. Barringer, and D. Rydeheard, "A pattern-based approach to parametric specification mining," in *Proc. 28th IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*, Nov. 2013, pp. 658–663.

[20] P. A. Brooks and A. M. Memon, "Automated GUI testing guided by usage profiles," in *Proc. 22nd IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*, Nov. 2007, pp. 333–342.

[21] A. Machiry, R. Tahiliani, and M. Naik, "Dynodroid: An input generation system for Android apps," in *Proc. ESEC/FSE 9th Joint Meeting Found. Softw. Eng.*, 2013, pp. 224–234.

[22] S. Anand, M. Naik, H. Yang, and M. J. Harrold, "Automated concolic testing of smartphone apps," in *Proc. ACM Conf. Found. Softw. Eng. (FSE)*, 2012, Art. no. 59.

[23] N. Mirzaei, S. Malek, C. S. Păsăreanu, N. Esfahani, and R. Mahmood, "Testing Android apps through symbolic execution," *ACM SIGSOFT Softw. Eng. Notes*, vol. 37, no. 6, pp. 1–5, Nov. 2012.

[24] C. Cadar, D. Dunbar, and D. Engler, "KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs," in *Proc. 8th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2008, pp. 209–224.

[25] P. Godefroid, N. Klarlund, and K. Sen, "DART: Directed automated random testing," in *Proc. ACM Conf. Program. Lang. Design Implement. (PLDI)*, 2005, pp. 213–223.

[26] J. C. King, "Symbolic execution and program testing," *Commun. ACM*, vol. 19, no. 7, pp. 385–394, Jul. 1976.

[27] R. C. Bryce, S. Sampath, and A. M. Memon, "Developing a single model and test prioritization strategies for event-driven software," *IEEE Trans. Softw. Eng.*, vol. 37, no. 1, pp. 48–64, Jan./Feb. 2011.

[28] A. M. Memon, M. E. Pollack, and M. L. Soffa, "Automated test oracles for GUIs," in *Proc. ACM Conf. Found. Softw. Eng. (FSE)*, 2000, pp. 30–39.

[29] A. M. Memon and M. L. Soffa, "Regression testing of GUIs," in *Proc. ACM Conf. Found. Softw. Eng. (FSE)*, 2003, pp. 118–127.

[30] X. Yuan, M. B. Cohen, and A. M. Memon, "GUI interaction testing: Incorporating event context," *IEEE Trans. Softw. Eng.*, vol. 37, no. 4, pp. 559–574, Jul./Aug. 2011.

[31] L. White and H. Almezen, "Generating test cases for GUI responsibilities using complete interaction sequences," in *Proc. 11th IEEE Int. Symp. Softw. Rel. Eng. (ISSRE)*, 2000, p. 110.

[32] X. Yuan and A. M. Memon, "Generating event sequence-based test cases using GUI runtime state feedback," *IEEE Trans. Softw. Eng*, vol. 36, no. 1, pp. 81–95, Jan./Feb. 2010.

[33] D. Amalfitano, A. R. Fasolino, S. De Carmine, A. M. Memon, and P. Tramontana, "Using GUI ripping for automated testing of Android applications," in *Proc. 27th Int. Conf. Autom. Softw. Eng. (ASE)*, 2012, pp. 258–261.

[34] T. Takala, M. Katara, and J. Harty, "Experiences of system-level model-based GUI testing of an Android app," in *Proc. 4th Int. Conf. Softw. Test., Verification Validation (ICST)*, 2011, pp. 377–386.
[35] W. Enck *et al.*, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proc. 9th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2010, pp. 393–407.
[36] P. Gilbert, B.-G. Chun, L. P. Cox, and J. Jung, "Vision: Automated security validation of mobile apps at app markets," in *Proc. 2nd Int. Workshop Mobile Cloud Comput. Services (MCS)*, 2011, pp. 21–26.
[37] L. K. Yan and H. Yin, "DroidScope: Seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis," in *Proc. 21st USENIX Secur. Symp.*, 2012, pp. 1–16.
[38] P. J. Phillips, H. Wechsler, J. Huang, and P. J. Rauss, "The FERET database and evaluation procedure for face-recognition algorithms," *Image Vis. Comput. J*, vol. 16, no. 5, pp. 295–306, 1998.
[39] P. J. Phillips, H. Moon, S. A. Rizvi, and P. J. Rauss, "The FERET evaluation methodology for face recognition algorithms," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 10, pp. 1090–1104, Oct. 2000.

DEBDEEP BANERJEE received the master's degree in electrical engineering from the Illinois Institute of Technology. He has approximately 10 years of industry experience in the field of software/systems engineering. He is currently a Senior Staff Engineer and an Engineering Manager with Qualcomm Technologies, Inc., USA. He is also a Software/Systems Development Engineer and the Test Lead for the Computer Vision Project. He is responsible for the test automation design, planning, development, deployment, code reviews, and project management. He has been with the Software Test Automation Team since the inception of the computer vision project with Qualcomm Technologies, Inc. He is involved in managing and developing software for the computer vision lab using the robotic arm.

KEVIN YU is currently a Test Engineer with Qualcomm Technologies, Inc., USA, and has contributed to the test automation validation for the continuous integration of computer vision algorithms. He has also validated computer vision engine features such as image rectification for the Android software products.