Hindawi
Mobile Information Systems
Volume 2022, Article ID 1084602, 13 pages
https://doi.org/10.1155/2022/1084602

Research Article

Learning-Replay Based Automated Robotic Testing for Mobile App

Feng Xue,[1] Junsheng Wu,[2] and Tao Zhang[2]

[1]*School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an 710072, China*
[2]*School of Software, Northwestern Polytechnical University, Xi'an 710072, China*

Correspondence should be addressed to Tao Zhang; tao_zhang@nwpu.edu.cn

Received 30 January 2022; Accepted 21 February 2022; Published 16 March 2022

Academic Editor: Hye-jin Kim

Record-replay testing is widely used in mobile app testing as an automated testing method. However, the current record-replay methods are closely dependent on the internal information of the device or app under test. Due to the diversity of mobile devices and system platforms, their practical use is limited. To break this limitation, this paper proposes an entirely black-box learning-replay testing approach by combining robotics and vision technology to achieve a record-replay testing that can support cross-device and cross-platform. Firstly, vision technology is used to extract the critical information of GUI and gesture actions during the tester's testing process; secondly, the GUI composition and test actions are analyzed to form a test sequence; finally, the robotic arm is guided to complete the replay of the test sequence through visual judgment. On the one hand, the approach in this paper does not access the interior of the app, shielding the association between test actions and device; on the other hand, it captures more abstract test action information instead of simple operation location records and supports more flexible test action replay. We demonstrate the effectiveness of this approach by evaluating the learning-replay of 12 popular apps for 13 typical scenarios on the same device, across devices, and across platforms.

1. Introduction

Mobile apps have shown a high-speed development trend in recent years. They have been gradually substituting desktop software to serve us. Compared with desktop software, mobile apps have new features such as rapid number growth, frequent version iteration, and diverse system platforms. However, these features of mobile apps present challenges for software testing both in workload and difficulty. In view of this, automated testing methods are more urgently needed for mobile apps to ensure quality [1].

Record-replay testing is a classic automated testing method. It records the tester's test actions, converts them into test scripts, and then uses the scripts to replay the test actions on the app under test (AUT) [2]. Record-replay plays a vital role in software testing, especially in regression testing that requires repeated verification. Industry and academia continue to make efforts to improve the performance of record-replay testing.

Event driven record-replay testing captures events of app to replay. Reran [3], a typical representative of record-replay testing, can accurately reproduce the test actions by capturing the underlying system event stream and can flexibly define the event-triggered time limit during replay. Reran obtains test information from the system level and can simulate nondiscrete actions, such as irregular sliding, but it is closely related to the device under test. Mosaic [4] and Appetizer [5] are similar to Reran. In addition, operating system platforms provide specialized testing frameworks or tools, such as Monkeyrunner [6] and Espresso [7] for Android app and XCTest [8] for iOS app, but they only work for testing apps on their own platform. Appium [9] implements cross-platform testing through script conversion by encapsulating the above testing frameworks of each platform. Calabash [10] also realizes cross-platform testing by converting test scripts for testing frameworks of different platforms. However, platform differences still make cross-platform testing difficult in this way. SARA [11] achieves a

high replay success rate for single-device and cross-device by binding events to Android widgets. Amalfitano [12] considered the acquisition of GUI layout file information in the process of record to correspond action events with GUI controls. In the subsequent replay, accurate event replay can be performed by searching for corresponding controls in a targeted manner. These works combine events with GUI layout to improve the accuracy of replay.

The GUI driven record-relay testing further captures GUI images to replay. With the adoption of computer vision in software engineering [13], GUI images are included in mobile app testing. Sikuli [14] and Eyeautomate [15] can generate scripts that contain screenshots of GUI elements and replay across devices by comparing image pixels of the elements. Based on this idea, Airtest [16] and Ranorex [17] utilize modern visual technology to generate such visual scripts and achieve higher accuracy replay. Furthermore, LIRAT [18, 19] tries to match image and layout characterization of GUI to solve the problem that the test cannot be replayed after fine-tuning due to version update or different platforms.

In the current record-replay testing techniques, an essential basis is that they all need to access the interior of the app to obtain either GUI information through layout files or test actions information through event streams. Thus, they are closely dependent on the system platform or AUT and cannot support cross-device and cross-platform testing well. However, mature mobile apps usually need to adapt to multiple mobile devices and multiple system platforms (e.g., iOS, Android, and Web). Differences in screen size, resolution, operating system type, and operating system version of mobile devices will lead to the failure of record and replay. In actual use, the maintenance workload for the record scripts to ensure the validity of the record-replay testing is still huge [20]. In addition, multiplatform version apps require independent record-replay testing tools for each platform, which will also increase the cost and complexity of testing.

At the same time, mobile apps require interaction with the outside world, so the traditional testing method using simulated signals to drive simulated events can no longer support the test coverage of mobile apps. Therefore, robotic testing for mobile apps is proposed to enhance the realism of the testing [21–23]. The industry has also launched various robotic arm based automated software testing platforms [24–26]. However, in the current automated robotic testing research, the robot is only used as an actuator, and a whole-process automated robotic testing approach has not been formed.

Regarding the issues above, this paper proposes a learning-replay based automated robotic testing approach that learns and imitates the test actions from a complete black-box perspective. We expect the robot to recognize and test apps from the external appearance, just like humans. The robot captures test information through vision technology and uses the robotic arm to implement test execution, realizing the decoupling of testing from devices and apps. Thereby, it can solve the shortcomings of the current record-replay testing in cross-platform and cross-device issues.

The main idea of the learning-replay based automated robotic testing is as follows: firstly, the tester's testing process is recorded as a video; secondly, vision technologies are used to visually capture the test information from the video, and then the test sequence is formed; finally, the test replay is completed by the robotic arm according to the test sequence automatically.

The purpose of this paper is to achieve an entirely automated black-box testing for mobile app by learning and mimicking human test action. It can identify what operations can be done on what elements under what kind of GUI structure. The approach not only is a record of testers' test actions, but can identify more information and achieve comprehensive judgment as follows:

(a) The ability to test due to GUI structure for cross-platform and cross-device testing.

(b) The ability to select test scripts autonomously and execute interactively.

2. Approach Description

Figure 1 shows an overview of our approach based on the automated robotic testing environment. The environment provides complete black-box testing without relying on any internal information of AUT. In detail, the camera captures the GUI and test actions to AUT, and the robotic arm imitates the interaction between tester and AUT. Like record-replay testing, our approach consists of a learning (record) phase and a replay phase.

In the learning phase, the camera is used to record the tester's testing process on AUT, and then vision technologies are utilized to identify test actions, including GUI and gesture information in each test step, thereby forming a test sequence. The goal of this phase is as follows: we expect the robot to learn the tester's testing intentions instead of rigidly recording the test.

In the replay phase, the robot first captures the GUI of AUT through the camera, then determines the executable action based on the learned test sequence, and finally drives the robotic arm to complete the action. The goal of this phase is as follows: we expect to achieve a visual-driven automated test execution process like humans.

The details of the learning-replay based automated robotic testing are given below. The framework of the learning-replay approach is shown in Figure 2. Moreover, we design algorithms for the test learning and test replay processes, respectively. Algorithm 1 describes the generation process of test sequence scripts in the learning phase. The time cost of Algorithm 1 is $O(n + mn)$, where n is the number of video frames, and m is the number of elements in a single GUI. Algorithm 2 describes the test execution process in the replay phase. The time cost of Algorithm 2 is $O(pq)$, where p is the number of AUT's GUI, and q is the total number of test scripts.

2.1. Video Recording. Video recording means recording the tester's testing process by video, as shown in Figure 3. Because the test environment only uses an upper monocular

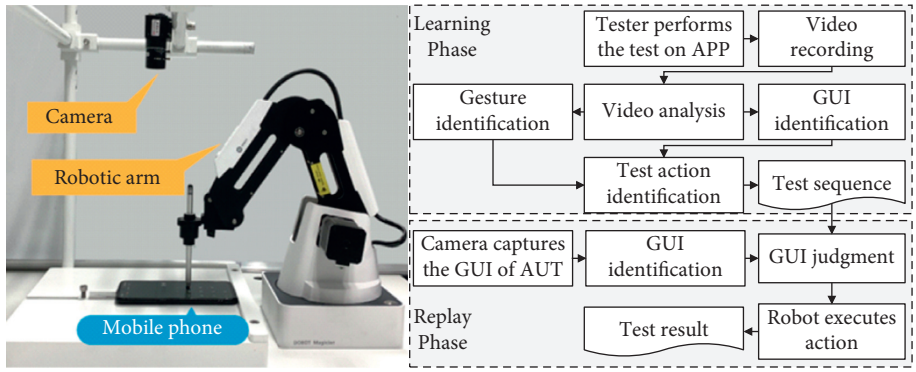

Figure 1: Learning-replay based automated robotic testing overview.

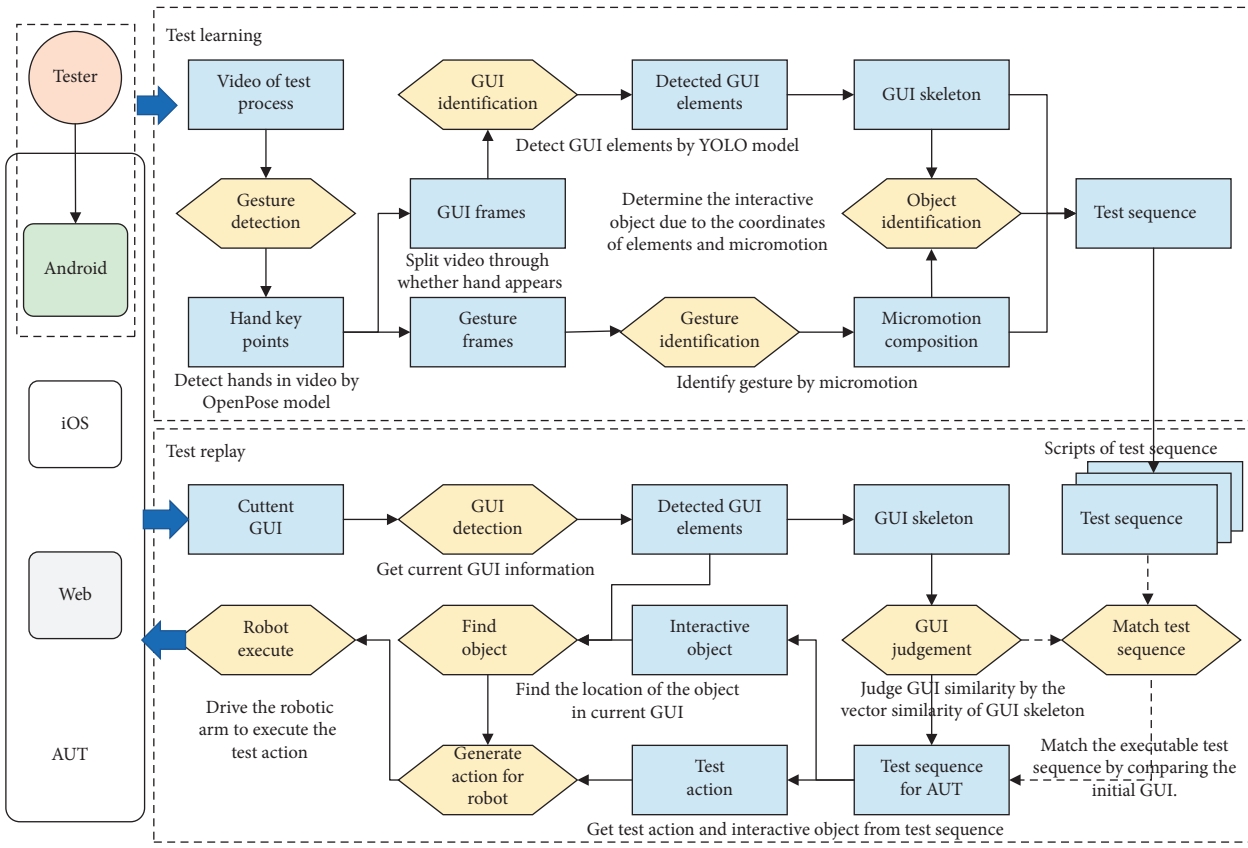

Figure 2: The framework of the learning-replay testing approach.

camera, during the test execution, the hand needs to enter the shooting area from outside and completely leave the area after completing a certain action. In addition, the shooting area needs to be able to capture the GUI and the hand movement trajectory. A recorded video is generated for each test case executed by the tester.

2.2. Video Analysis. When the recorded video is obtained, it is necessary to further segment the GUI frames and the gesture action frames in the video. Here, we introduce OpenPose [27], a model that can detect the 2D pose of people in real time, and it is used to detect hand trajectories in the video.

As shown in Figure 4, OpenPose can output the key points of the hand in each video frame. From this, we can extract the fingertip trajectory of the index finger (the finger that touches the screen) for each test action in the video.

Therefore, the frames where the fingertip trajectory continuously appears are the gesture action frames, and the frames where the hand is not detected are the GUI frames; Figure 5 gives an example. Figure 5 shows the coordinate change value of the fingertip position in each frame of the video. Frames with coordinate change value (i.e., the hand is detected) are gesture frames. Frames without coordinate change value (i.e., no hand is detected) are GUI frames. And the GUI frames before and after the gesture action frames,

```
(i) Input: recorded video RV
(ii) Output: test sequence script TSS
(1) split_tag = 0
(2) for each frame in RV do
(3)         finger = getOpenPoseDetection(RV)
(4)         if finger is exist then
(5)                 gesture_info ⇐ map(frame, split_tag, finger)
(6)         else
(7)                 gui_info ⇐ map(frame, split_tag)
(8)                 if pre_frames is gesture frame then
(9)                         split_tag ++
(10)                end if
(11)        end if
(12) end for
(13) for st = 0 to split_tag do
(14)        gui = getMiddleFrame(gui_info, st)
(15)        gui_elements = getObjectDetection(gui)
(16)        gui_skeleton = getGuiSkeleton(gui_elements)
(17)        gesture_micro = identifyGesture(gesture_info, st)
(18)        object = identifyObject(gesture_micro, gui_elements)
(19)        TTS ⇐ getTestSquence(st, gui_skeleton, gesture_micro, object)
(20) end for
(21) return TSS
(22) function identifyGesture(gesture_info, st)
(23)        t = getInterMicroFrames(gesture_info,st,f_threshold)
(24)        if |coord_firstframe(t) − coord_endframe(t)| < c_threshold then
(25)                return map(move, coord_first, coord_end)
(26)        else if length(t) < n_threshold then
(27)                return map(none, coord_first, null)
(28)        else if length(t) > h_threshold then
(29)                return map(hold, coord_first, null)
(30)        else
(31)                return map(tap, coord_first, null)
(32)        end if
(33) end function
(34) function identifyObject(gesture_micro, gui_elements)
(35)        for each element in gui_elements do
(36)                isContain = contain(coord_first, element)
(37)                if isContain = true then
(38)                        return element
(39)                end if
(40)        end for
(41)        return surface
(42) end function
```

Algorithm 1: Generation of test script

respectively, represent the initial GUI before each action, and the response GUI after the action is executed. We add split tags to each frame by judging the transition of the GUI frames and gesture frames (lines 4–11 in Algorithm 1).

2.3. GUI Identification and Judgement. For the automated robotic testing, we expect the robot can learn what GUI element can be operated under what GUI structure. The preliminary work [28] we have done is to realize the visual identification of GUI elements by training the object detection model (YOLO [29, 30]) and abstract the GUI as a GUI skeleton composed of control elements. Each control element contains its category and bounding box information. Further, we have built a deep learning model with autoencoder architecture [31] to extract the feature vectors of the GUI skeleton and take relative entropy to measure the difference of vectors to determine the similarity between GUIs. In this paper, we expand the categories of detected GUI elements to 10 (including TextView, EditText, Button, Switch, RadioButton, CheckBox, ImageView, ImageButton, SeekBar, KeyBoard) to enhance the capture of GUI information. Figure 6 shows the visual detection of GUI elements and the construction of the GUI skeleton.

The construction and comparison of GUI skeletons are necessary. The image that needs to be processed is a camera shooting area but not just a GUI image. Pure image pixel comparison is ineffective. We expect that the robot can focus

```
(i) Input: test sequence scripts TSSs, AUT
(ii) Output: test result TR
(1) serial_number = 1
(2) while true do
(3)         gui_current = getCurrentGUI()
(4)         gui_elements' = getObjectDetection(gui_current)
(5)         gui_skeleton' = getGUISkeleton(gui_elements')
(6)         if serial_number is 0 then
(7)             break
(8)         else if serial_number is 1 then
(9)             for each tss in TSSs do
(10)                gui_skeleton = getTSSGUI(tss, serial_number)
(11)                e = getSimilarityJudgment(gui_skeleton', gui_skeleton)
(12)            end for
(13)            TSS' = getMaxSimliarScript(e, s_threshold)
(14)            serial_number = exectеAction(TSS', serial_number, gui_elements')
(15)        else
(16)            gui_skeleton = getTSSGUI(TSS′, serial_number)
(17)            e = getSimilarityJudgment(gui_skeleton', gui_skeleton)
(18)            if e > s_threshold then
(19)                TR ⇐ recordResults()
(20)                serial_number = exectueAction(TSS', serial_number, gui_elements')
(21)            else
(22)                TR ⇐ recordResults()
(23)                break
(24)            end if
(25)        end if
(26) end while
(27) return TR
(28)
(29) function executeAction(TSS', serial_number, gui_elements')
(30)        object = getTTSObject(TSS′, serial_number)
(31)        object' = findObject(gui_elements', object)
(32)        if object' is find then
(33)            gesture' = getGesture(TSS′, serial_number)
(34)            action = generateAction(object', gesture')
(35)            executeRoboticArm(action)
(36)            serial_number = step(serial_number)
(37)        else
(38)            serial_number = 0
(39)        end if
(40)        return serial_number
(41) end function
```

Algorithm 2: Test replay

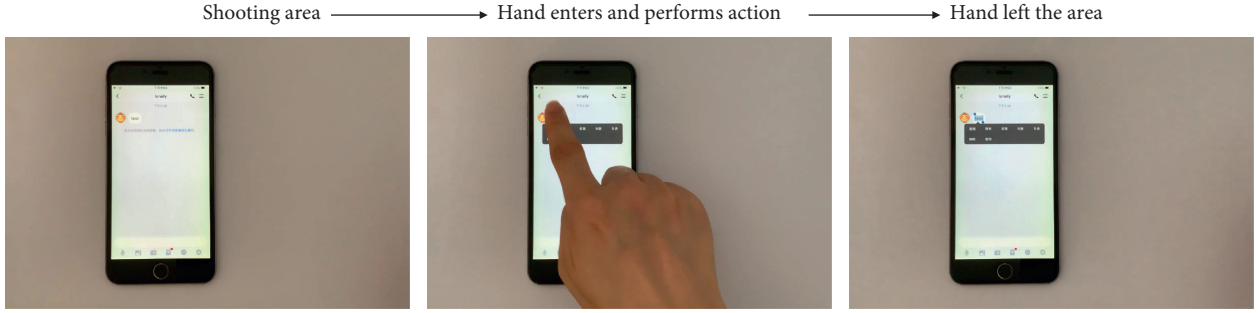

Figure 3: Record the tester's testing process with video.

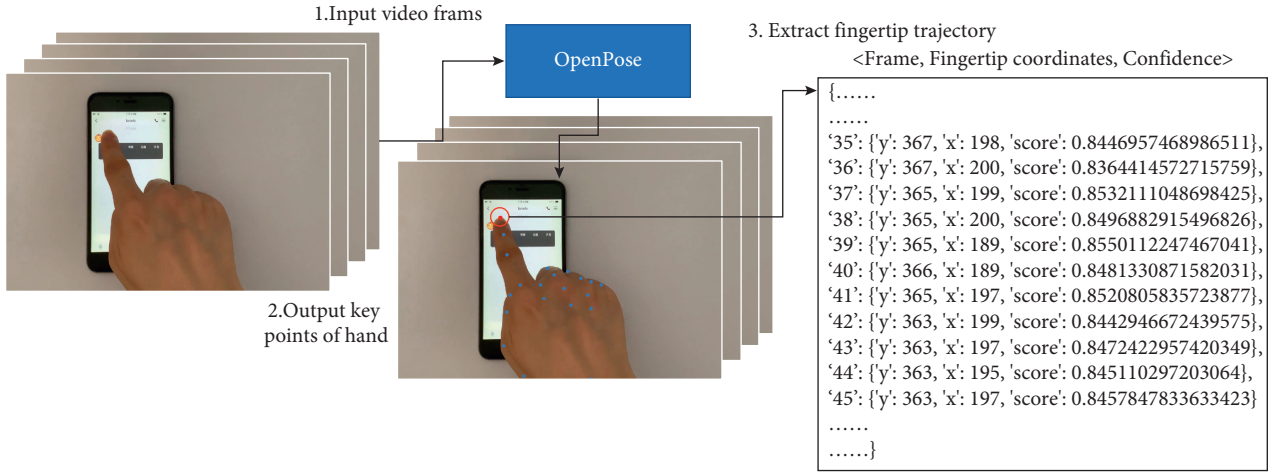

Figure 4: Detect and extract fingertip trajectory through OpenPose model.

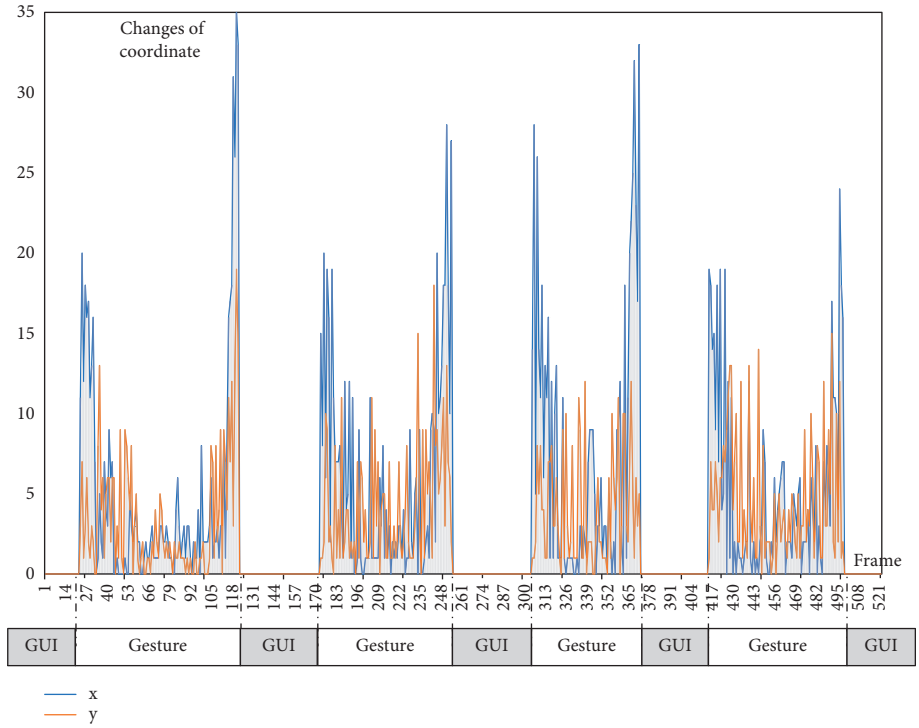

Figure 5: Determine GUI frames and gesture frames through fingertip trajectory.

on the GUI structure while shielding the interference information, such as changes in the position of the mobile phone, different element sizes, and different element styles of AUT.

2.4. Test Gesture Identification. For the test actions, we expect the robot to learn what actions the tester did rather than just record the action trajectory. We do not directly judge the action but identify the micromotion of gesture action. Refer to the standard gesture actions supported in the development instructions of iOS [32] and Android [33], as shown in Table 1 (here, we only consider the frequently used single-finger actions), which can all be represented by micromotions.

Tap is a click action that triggers a control by clicking on a control element in the GUI. Long press a control element to activate its function, such as a long press on a text to trigger the copy menu. Double tap a control element, such as double tap a picture to zoom in or out. Flick is a quick touch and movement of the surface, such as scrolling the surface vertically to browse all the contents in a list. Drag is to hold down an element or surface and move it, such as moving an element to another location or sorting items in a list. Swipe is to slide the control element or surface to the left or right,

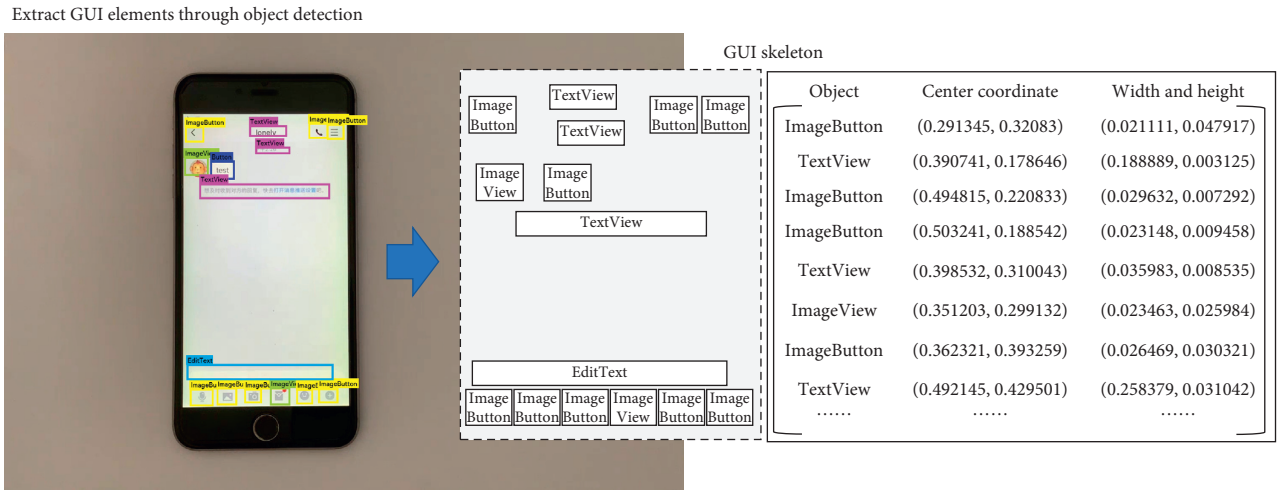

Figure 6: Detect GUI elements and construct GUI skeleton.

Table 1: Gesture action.

Standard gestures	Interactive object	Micromotion composition
Tap	Elements	<touch, none, leave>
Long press (touch and hold)	Elements	<touch, hold, leave>
Double tap	Elements	<touch, touch, leave>
Scroll and pan (flick)	Surfaces	<touch, move, leave>
Drag	Surfaces/Elements	<touch, move, leave>
Swipe	Surfaces/Elements	<touch, move, leave>

such as switching tabs or removing an item from a list by sliding horizontally.

The interactive objects of all actions contain elements and surfaces. Some gestures are only valid for elements, some for surfaces, and some for both. Gestures against surfaces tend to be more guided, while gestures against elements have both guided and behavioral implications. In general, conflicting gestures are not recommended for iOS and Android designs [32, 33].

Analysis of the characteristics of these gesture actions reveals that all of them can be composed of the four micromotions: touch, move, hold, and leave. Touch is the initial sign of all gesture actions, and leave is the end sign. When the tester's hand moves towards the interactive object, it presents a speed change from fast to slow and finally appears to be relatively static for multiple frames when the action occurs and then accelerates away. Taking Figure 7 as an example, we calculate the coordinate change value for each frame of a long press action. The coordinate change is almost maintained at a relatively low position when the long press action occurs. To trigger an action, the finger needs to contact the screen for a certain period. The length of this period can determine the type of the intermediate micromotion (touch time requirement: none < touch < hold). Moreover, according to the change of the contact area before and after the period, it is determined whether a move micromotion occurs.

Therefore, we first calculate the fingertip position detection error when the finger is in a continuous static state as the threshold for judging whether the finger is in contact with the screen. If there is a constant frame segment lower than the threshold, it is determined to be the period when the finger touches the screen, that is, the intermediate micromotion frames. Then, we evaluate the average frame number of intermediate micromotion for tap, long press, and double tap gestures as the measure threshold for none, hold, touch, and the average deviation of fingertip position within the intermediate micromotion frames as the measure threshold for move. Thus, the test gesture identification is completed.

The gesture identification process is described as function identifyGesture in Algorithm 1 (line 22–33). For our robotic testing environment, f_threshold is set as 7.1 to judge and extract frames where the intermediate micromotion occurs, c_threshold is set as 15 to judge the change of fingertip position between the first and last frame during the intermediate micromotion occurs, n_threshold is set as 42 to judge whether it is none micromotion, and h_threshold is set as 125 to judge whether it is hold micromotion. For the nonmove micromotions, the function will return the micromotion category and the first frame fingertip coordinate; for the move micromotion, the function will return the category and the fingertip coordinates of the first frame and end frame.

2.5. Test Action Identification. After the test gesture and GUI information are extracted from the video, it can be inferred what kind of action the tester performed on an interactive object due to the contact position when the action occurs.

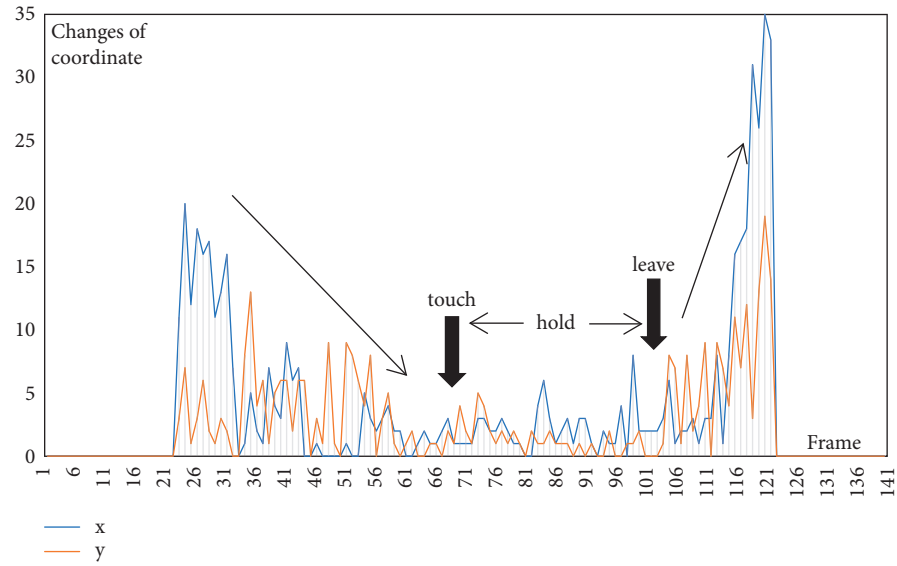

Figure 7: The coordinate changes of each frame of a long press action.

Then, the generated test sequence is defined as [Serial number, GUI skeleton, Micromotion decomposition, interactive object]. Serial number denotes sequence order, and it will be organized into a tree structure if there are multiple videos for a single app. GUI skeleton is constructed in Section 2.3. Micromotion decomposition is generated in Section 2.4. Interactive object is determined by the object detection results of GUI elements in Section 2.4 and the fingertip position of gesture in Section 2.5. The function identifyObject defines the judgment of the interactive object (lines 34–42 in Algorithm 1). If the coordinates of the fingertip fall in the bounding box of a GUI element, it is the element; else, it is the surface. The test sequence script is shown in Figure 8.

The script uses a more advanced form of expression instead of recording the underlying location information to better support cross-platform testing.

2.6. Test Replay. In the replay phase, we do not specify the test sequence to be executed but let the robot determine the executable test based on the learned test sequence. The detailed replay process is as follows:

(a) At the beginning of the test, the initial GUI of AUT is randomly placed in the camera shooting area. After the robot captures the current GUI, it matches the GUI with the initial GUI in the test sequence, looking for possible test action (lines 6–13 in Algorithm 2). Here, the method of comparing the vector similarity of two GUI skeletons in Section 2.4 is adopted to match the GUI. For our robotic testing environment, s_threshold is set as 0.85.

(b) The robot locates the interactive object on the current GUI according to the interactive object and the test gesture (micromotion decomposition) defined by the test sequence. Then, it drives the robotic arm to complete the action execution. The function executeAction describes this process (lines 29–41 in Algorithm 2).

(c) After completing the test action, the robot obtains the response GUI of AUT. First, the robot determines whether the interface has changed, that is, whether the test action is executed successfully; second, it determines whether the current GUI matches the GUI in the next sequence defined by the test sequence, that is, whether the response of AUT meets the expectations. If it meets the expectations, execute the next sequence; otherwise, report an exception (lines 15–25 in Algorithm 2).

[1,	GUI0,	<touch, none, leave>,	ImageView4]
[1.1,	GUI1,	<touch, hold, leave>,	Button2]
[1.1.1,	GUI2,	<touch, none, leave>,	TextView8]

Figure 8: The test sequence script [Serial number, GUI skeleton, Micromotion decomposition, interactive object].

Repeat the above process until the completion of a test sequence. Test result records will be generated whether the test replay is successful or not. During the replay process, the test will be executed by the robot in real time interactive judgment, rather than rigid action playback.

3. Experiment

To evaluate the effectiveness of our approach, we perform a study of testing popular apps under different platform versions and devices.

3.1. Experiment Setup. We select 6 mobile devices with different screen sizes, resolutions, and operating systems and install popular apps on the iOS and Android App Stores, covering categories such as social, communication, entertainment, and news. Details are shown in Tables 2 and 3.

Table 2: Devices for the experiment.

ID	Device	Operating system	Screen Size(inch)	Resolution
I1	Apple iPhone 7 plus	iOS 14.2	5.5	1920×1080
I2	Apple iPad mini 4	iPadOS 13.5	7.9	2048×1536
A1	Huawei Honor v10	Android 8.1	5.99	2160×1080
A2	Samsung Galaxy Tab S6	Android 9.0	10.5	2560×1600
W1	Xiaomi Mi Max 3	Web	6.9	2160×1080
W2	Apple iPhone 8	Web	4.7	1334×750

Table 3: Apps for the experiment.

App name	Categories	Install	Supported platforms	Device
Reddit	Social	50m+	iOS/Android/Web	I1
WeChat	Communication	100m+	iOS/Android	I1
Google translate	Tools	1000m+	iOS/Android/Web	I2
Amazon shopping	Shopping	100m+	iOS/Android/Web	I2
Zoom	Business	500m+	iOS/Android	A1
Opera news	News	100m+	iOS/Android/Web	A1
Booking.com	Travel	100m+	iOS/Android/Web	A2
Wikipedia	Books	50m+	iOS/Android/Web	A2
Outlook	Productivity	500m+	iOS/Android/Web	W1
The weather channel	Weather	100m+	iOS/Android/Web	W1
Apple music	Music	50m+	iOS/Android/Web	W2
YouTube	Video	5000m+	iOS/Android/Web	W2

In terms of recording the test video, we write test cases for each app around the 9 typical scenarios, including select, search, login, forward, comment, open, add to, setting, zoom-out, browse, hidden menu, tabs-switch, and move, and then assign multiple testers (senior graduate students) to complete the execution of the test cases. All testing procedures are recorded by the camera. The scenarios' distribution of apps is shown in Table 4.

After the recorded videos are obtained, the learning phase of the robotic testing begins. The robot learns the test actions from the videos and generates test sequence scripts. Then, the replay phase of the robotic testing can be conducted to verify the validity of the learning-replay testing.

In the replay phase, three groups of experiments are carried out: replay on the original device, relay on the original device that recorded the test video; replay across devices of the same platform, replay between devices of the same system platform; and replay across devices and platforms, replay between devices of different system platforms.

The success rate (SR) is the metric to evaluate the correctness of the learning-replay testing, which is denoted as

$$\mathrm{SR} = \frac{P_{\mathrm{actual}}}{P_{\mathrm{true}}}. \quad (1)$$

where P_{actual} represents the actual result, and P_{true} represents the ideal result. In the testing experiment, the test scripts generated by each app are replayed in the robotic testing environment, each test script is executed once, and the SR of each test action execution is calculated.

3.2. Results and Discussion. The results of the experiment are shown in Figure 9. TP, LP, DT, FK, DG, and SP represent tap, long press, double tap, flick, drag, and swipe gestures, respectively. The vertical SR calculates the replay success of each app, and the horizontal SR calculates the replay success of each gesture.

3.2.1. Replay on the Original Device. For the replay on the original device, the average SR reached 93.9%. WeChat and Zoom even achieved 100% replay SR; namely, all test actions are successfully replayed. The poor performing Amazon Shopping and Booking.com also reached more than 85%. For the replay performance of test actions, except the SR of long press and drag, which are lower than 90%, the rest of the actions are over 90%.

The replay on the original device is relatively simple, and there is no need to consider differences in devices, platforms, and apps. Therefore, the replay SR of the original device is only affected by the effect of our proposed approach.

In terms of visual identification for apps, too complex GUI elements can easily cause confusion in GUI identification. For example, there are many videos or pictures in YouTube and Reddit, and the text or logos appearing in the videos or pictures will interfere with GUI identification. The design of some GUI will confuse object detection, such as Booking.com, which uses a lot of text, and some texts are buttons or links that can be clicked. The variability of the GUI causes the action to fail to execute, such as Reddit or Opera News, which will constantly update the content, resulting in changes in the GUI structure. Their browse lists usually consist of plain text, a combination of image and text, or a video.

In terms of executing actions, some actions have a large contact area and high operating tolerance, such as flick and

Table 4: The distribution of scenarios for apps.

	Select	Search	Login	Forward	Comment	Open	Add to	Setting	Zoom-out	Browse	Hidden menu	Tabs-switch	Move
Reddit		√			√	√		√		√			
WeChat	√	√		√	√	√		√	√	√	√		√
Google translate	√				√	√		√		√			
Amazon shopping	√	√	√	√		√	√	√	√	√	√	√	√
Zoom			√		√	√		√		√			
Opera news		√		√		√	√	√	√	√		√	√
Booking.com	√	√	√			√	√	√	√	√			√
Wikipedia		√				√	√	√		√			
Outlook	√	√		√	√	√	√	√		√	√		
The weather channel		√		√		√		√		√		√	
Apple music		√	√			√				√	√	√	
YouTube		√		√	√	√	√	√		√		√	

App	Replay on the original device							Replay on the same platform devices							Replay on cross-platform devices					
	TP	LP	DT	FK	DG	SP	*SR*	TP	LP	DT	FK	DG	SP	*SR*	I→A	I→W	A→I	A→W	W→I	W→A
Reddit	22/23	1/1	2/3	12/12	–	–	94.9%	21/23	1/1	3/3	12/12	–	–	94.9%	87%	75%	–	–	–	–
WeChat	43/43	2/2	1/1	20/20	3/3	2/2	100%	42/43	2/2	1/1	19/20	3/3	2/2	98%	85.2%	–	–	–	–	–
Google Translate	31/32	–	–	10/10	–	5/5	97.9%	32/32	–	–	10/10	–	5/5	96.2%	81.3%	65.2%	–	–	–	–
Amazon Shopping	46/54	3/3	2/2	15/18	–	5/6	85.5%	45/54	2/3	2/2	16/18	–	5/6	84.3%	–	–	56.4%	73.4%	–	–
Zoom	10/10	–	–	2/2	1/1	2/2	100%	10/10	–	–	2/2	1/1	2/2	100%	92.2%	–	–	–	–	–
Opera News	26/27	3/4	2/2	13/15	3/4	13/13	92.3%	25/27	3/4	2/2	13/15	3/4	12/13	89.3%	–	–	75.9%	43.0%	–	–
Booking.com	28/34	3/4	1/1	20/22	–	–	85.2%	26/34	4/4	0/1	20/22	–	–	84.7%	–	–	89.2%	68.5%	–	–
Wikipedia	36/37	2/2	–	8/8	–	–	97.9%	36/37	1/2	–	7/8	–	–	93.6%	–	–	81.1%	12.3%	–	–
Outlook	60/65	1/1	2/2	12/12	1/1	6/6	94.3%	58/65	0/1	1/2	12/12	1/1	6/6	89.6%	–	–	–	–	82.2%	83.6%
The weather channel	13/15	1/1	2/2	5/5	–	2/2	92.0%	12/15	1/1	2/2	4/5	–	2/2	84.0%	–	–	–	–	65.2%	56.7%
Apple music	27/28	2/2	1/1	5/5	–	5/5	97.6%	26/28	2/2	1/1	5/5	–	5/5	95.1%	78.3%	52.4%	–	–	–	–
YouTube	11/12	2/3	1/1	8/9	–	3/3	89.3%	12/12	2/3	1/1	8/9	–	3/3	92.8%	–	–	–	–	87.6%	85.4%
SR	98.9%	87.0%	93.3%	94.2%	88.9%	97.7%	93.9%	91.1%	78.3%	86.7%	92.8%	88.9%	95.4%	91.9%	84.8%	64.2%	75.7%	49.3%	78.3%	75.2%

Figure 9: The results of the replay experiment of the learning-replay testing. TP (Tap), LP (Long press), DT (Double tap), FK (Flick), DG (Drag), SP (Swipe), SR (Success rate); I (iOS), A (Android), W (Web), I⟶A (learning on iOS, replay on Android).

sliding. On the contrary, the tap fails due to inaccurate operation in some narrow areas and errors caused by recognition.

3.2.2. Replay on the Same Platform Devices. For the replay on the cross-device of the same platform, the SR of robotic test execution reached 91.9%, only slightly lower than the

Table 5: The success rate of learning and replay in each functional scenario.

	Select	Search	Login	Forward	Comment	Open	Add to	Setting	Zoom-out	Browse	Hidden menu	Tabs-switch	Move
Test cases	10	12	4	8	9	84	26	18	8	97	14	22	8
SR of learning	80%	83.3%	100%	87.5%	66.7%	82.1%	88.4%	94.4%	100%	83.5%	100%	100%	100%
SR of replay	87.5%	70%	50%	71.4%	83.3%	81.5%	91.3%	58.8%	100%	88.9%	92.9%	100%	75%

original device replay. Cross-device replay on the same platform uses the same app of the same platform, so there are only differences in properties of devices. (In the experiment, the tablet is still in portrait mode.)

Compared with the original device replay results, the SR of cross-device replay on the same platform has almost all dropped. Changes in resolution or screen size still affect the recognition of GUI elements, especially from high-resolution devices to low-resolution devices.

In addition, an unexpected result is that the SR of a few actions is higher than that of the original device replay. Two reasons cause this result. One is that our approach abstracts the test actions into a test script, so the execution of the test is independent of the original device. Another is that the adopted visual detection algorithm has an inevitable detection error fluctuation. Changes in the accuracy of the detection box size can influence the execution results. Such subtle fluctuations may impact the test accuracy. However, in practice, multiple rounds of testing can be used to eliminate this effect.

It can be seen from the replay results that the execution SR error is low, and the proposed approach can effectively support replay across devices of the same platform.

3.2.3. Replay on Cross-Platform Devices. In the cross-platform replay, the test replay execution between iOS, Android, and Web platforms is completed, respectively. Compared with the first two sets of experiments, the cross-platform test results of each app have dropped significantly. Android to iOS, iOS to Android, Web to Android, and Web to iOS achieved SR between 75% and 85%. However, the SR of iOS to Web and Android to Web are only 64.2% and 49.3%.

We find that the main reasons for the decline of SR are as follows:

(i) First, it is undeniable that the GUI design of cross-platform apps is slightly different due to the differences of platforms. Especially between the web platform and the mobile system platform, the web is more about the display adaptation on the screen size of the mobile phone as well as the adaptation of ordinary gestures (click, flick) but does not consider the adaptation of uncommon gestures. For example, for the swipe gesture, the tab pages can be switched by swipe on the mobile system platform but clicking the tab to switch page is more used on the web platform. Among the experimental apps, only Apple Music's web version supports swipe to switch the tab pages.

(ii) Second, some apps use a hybrid development approach, nesting the web within the mobile app. For example, for Amazon Shopping, the browsing area of Android and Web is the same but different from iOS. The navigation buttons outside the browsing area are different for each platform.

(iii) Third, the web version of some apps is entirely different from the mobile version, such as Wikipedia.

Despite the drop in SR, cross-platform replay of more than half of the test actions is still achieved through the abstraction ability of our approach. To the best of our knowledge, this is the first time that record-replay testing has been applied to replay on multiple platforms from a black-box perspective. To a certain extent, we eliminate the impact of device and app differences on testing, thereby supporting record-replay across platforms.

The learning SR and replay SR of each functional scenario are shown in Table 5. It demonstrates that our approach can support test execution in various scenarios. However, some scenarios requiring sequence actions to complete will affect the SR, such as login and settings. Because of the sequential nature of test actions, the failure to execute some key actions will lead to the failure of subsequent actions to be executed.

4. Conclusion

In this paper, aiming at the shortcomings of current record-replay testing in cross-platform and cross-device testing, we propose a learning-replay testing approach based on robot vision. Our approach extracts the key information of GUI and gesture action in the testing process through vision technology and then transforms it into a test sequence, driving the robotic arm to complete the replay of test actions. The results of the experiment on popular apps and multiple scenarios prove that this approach has the ability of cross-device and cross-platform testing for black-box.

There are a large number and various types of mobile apps. We only use limited devices and apps for the verification of our approach. Thus, we may not have verified all the conditions. However, we have tried our best to collect representative apps, devices, and typical scenarios to participate in our experiment to reduce this impact.

There are still some limitations in this work. First, some apps that require sensor signals (e.g., gravity sensing) still challenge this entire black-box testing. It will put higher requirements on the robotic test environment and supporting algorithms. Second, most game apps have relatively independent characteristics of GUI and interaction, which will lead to the failure of our approach. Third, we only

implemented the recognition and replay of single-finger gestures, while multifinger actions are still used in some scenarios.

In the future, we will continue to improve this approach and promote its integration with the commercial mobile apps testing process. In addition, the essence of our approach is to judge the GUI structure to complete replay, but how to replay according to the meaning of GUI is the key to solving the cross-platform testing and even the cross-app testing of the same functional scenario, which will further enhance the automation of mobile app testing.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no potential conflicts of interest.

References

[1] L. Cruz, R. Abreu, and D. Lo, "To the attention of mobile software developers: guess what, test your app!" *Empirical Software Engineering*, vol. 24, no. 4, pp. 2438–2468, 2019.

[2] W. Lam, Z. Wu, D. Li, and W. Wang, "Record and Replay for Android: Are We There yet in Industrial cases?" in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pp. 854–859, Paderborn, Germany, August 2017.

[3] L. Gomez, I. Neamtiu, T. Azim, and T. Millsten, "Reran: Timing-And Touch-Sensitive Record and Replay for android," in *Proceedings of the 2013 35th International Conference on Software Engineering (ICSE)*, pp. 72–81, IEEE, September 2013.

[4] M. Halpern, Y. Zhu, and R. Peri, "Mosaic: Cross-Platform User-Interaction Record and Replay for the Fragmented Android ecosystem," in *Proceedings of the 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 215–224, IEEE, Philadelphia, PA, USA, March 2015.

[5] Appetizer Replaykit, "Control Multiple devices," 2022, https://github.com/appetizerio/replaykit.

[6] Monkeyrunner, "Monkeyrunner," 2022, https://developer.android.com/studio/test/monkeyrunner.

[7] Espresso, "Espresso," 2022, https://developer.android.com/training/testing/espresso.

[8] XCTest, "UI tests for your Xcode project," https://developer.apple.com/documentation/xctest.

[9] Appium, "Automation for Apps," 2022, https://appium.io.

[10] Calabash, "Automated Acceptance Testing for Mobile Apps," 2022, https://github.com/calabash/calabash.

[11] J. Guo, S. Li, and J. G. Lou, "Sara: self-replay augmented record and replay for Android in industrial cases," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pp. 90–100, Beijing China, July 2019.

[12] D. Amalfitano, V. Riccio, N. Amatucci, V. D. Simone, and A. R. Fasolino, "Combining automated gui exploration of android apps with capture and replay through machine learning," *Information and Software Technology*, vol. 105, pp. 95–116, 2019.

[13] M. Bajammal, A. Stocco, D. Mazinanian, and A. Meshbah, "A survey on the use of computer vision to improve software engineering tasks," *IEEE Transactions on Software Engineering*, 2020.

[14] T. H. Chang, T. Yeh, and R. C. Miller, "GUI Testing Using Computer vision," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1535–1544, 2010.

[15] Eyeautomate, "Visual Script Runner," 2022, https://eyeautomate.com/eyeautomate.

[16] Airtest, "Cross Platform UI Automation IDE," 2022, https://airtest.netease.com.

[17] Ranorex and Ranorex Studio, "Functional UI Test Automation," 2022, https://www.ranorex.com.

[18] S. Yu, C. Fang, and Y. Feng, "Lirat: Layout and Image Recognition Driving Automated mobile Testing of cross-platform," in *Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1066–1069, IEEE, San Diego, CA, USA, November 2019.

[19] S. Yu, C. Fang, and Y. Yun, "Layout and image recognition driving cross-platform automated mobile testing," in *Proceedings of the 2021 IEEE/ACM 43rd international conference on software engineering (ICSE)*, pp. 1561–1571, Madrid, ES, May 2021.

[20] M. Nass, E. Alégroth, and R. Feldt, "Why many challenges with GUI test automation (will) remain," *Information and Software Technology*, vol. 138, Article ID 106625, 2021.

[21] M. Craciunescu, S. Mocanu, and C. Dobre, "Robot Based Automated Testing Procedure Dedicated to mobile devices," in *Proceedings of the 2018 25th International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 1–4, IEEE, Maribor, Slovenia, June 2018.

[22] K. Mao, M. Harman, and Y. Jia, "Robotic testing of mobile apps for truly black-box Automation," *IEEE Software*, vol. 34, no. 2, pp. 11–16, 2017.

[23] D. Banerjee, K. Yu, and G. Aggarwal, "Image rectification software test automation using a robotic ARM," *IEEE Access*, vol. 6, pp. 34075–34085, 2018.

[24] Sastra Robotics, "Automated HMI Testing Platforms," 2022, https://sastrarobotics.com/products.

[25] OptoFidelity, "Test Automation Capabilities," 2022, https://www.optofidelity.com.

[26] Quality Commander, "Automated Testing of Embedded Software," 2022, https://www.jnovel.co.jp/en/service/qc/#equipment.

[27] Z. Cao and T. Simon, "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 1, pp. 172–186, 2019.

[28] T. Zhang, Y. Liu, J. Gao, L. P. Gao, and J. Cheng, "Deep learning-based mobile application isomorphic GUI identification for automated robotic testing," *IEEE Software*, vol. 37, no. 4, pp. 67–74, 2020.

[29] J. Redmon, S. Divvala, and R. Girshick, "You only look once: unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 779–788, Las Vegas, NV, USA, JunE 2016.

[30] J. Redmon and A. Farhadi, "Yolov3: An Incremental improvement," 2018, https://arxiv.org/abs/1804.02767.

[31] K. Ghasedi Dizaji, A. Herandi, and C. Deng, "Deep clustering via joint convolutional autoencoder embedding and relative

entropy minimization," in *Proceedings of the IEEE international conference on computer vision*, pp. 5736–5745, Venice, Italy, October 2017.

[32] Human Interface Guidelines, "Gestures," 2022, https://developer.apple.com/design/human-interface-guidelines/ios/user-interaction/gestures.

[33] Material Design, "Gestures," 2022, https://material.io/design/interaction/gestures.html.