IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 11, 2018, accepted September 25, 2018, date of publication October 1, 2018, date of current version October 25, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2873284

Object Tracking Test Automation Using a Robotic Arm

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

ABSTRACT Touch-to-track is a software feature that is used to track objects in embedded systems, such as mobile phones. The research question that is discussed in the paper is as follows: how can one design an automated test method for accurately testing object tracking algorithms? The challenge for the verification team was to design a method of test automation for testing this object tracking feature. The solution to the problem was to develop test automation algorithms using a robotic arm to accurately test this software feature. The robotic arm is used for executing test cases for tracking single or multiple objects in motion. To use the touch-to-track feature, the user selects an area in the camera preview by drawing a bounding box. Hough transformation and color segmentation are applied to accurately detect the bounding box drawn on every frame in which the object is detected. The area inside the bounding box is considered as the template. Template matching algorithms based on normalized cross-correlation, phase correlation, and speeded up robust features (SURF)-based feature extraction and matching are then applied to compare the template and original images. The tracking efficiency is calculated by dividing the total number of frames that contain the object being tracked by the total number of frames being tested. The tracking efficiencies achieved with the Hough-based bounding box detection algorithm followed by template matching algorithms based on normalized cross-correlation, phase correlation, and SURF-based feature extraction and matching are compared. These tracking efficiencies are calculated by comparing the ground truth to the tracking results for each frame. The tracking efficiency calculated for three objects is 87.7% with no colored background. The proposed object tracking solution with hardware acceleration is found to perform better than the third-party solution with respect to the latency in re-establishing tracking and the hand jitter scenario.

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

I. INTRODUCTION

This technical paper focuses on the topic of test automation for the computer vision feature known as touch-to-track.

Object tracking is an extremely challenging task on embedded devices, which have stringent memory constraints. The computer vision algorithms that are used for tracking are typically computation-intensive. Therefore, implementations of real-time tracking can involve hardware acceleration using hardware units such as digital signal processors (DSPs).

Several key scenarios have been considered in the design and development of a method for object tracking test automation using a robotic arm. These key scenarios are listed below.

A. OBJECT OCCLUSION DURING TRACKING USING A ROBOTIC ARM

The robotic arm is programmed to track a moving test object. Screens are added along the path of the moving object to occlude the object for specific lengths of time. The time needed for the object tracking algorithm to begin tracking the object again after it emerges from such an occlusion is then calculated. This scenario is helpful for testing the efficiency of the software in restarting tracking after the object has passed out of the field of view (FOV).

B. HAND JITTER INDUCED DURING OBJECT TRACKING USING A ROBOTIC ARM TO SIMULATE REAL-WORLD USER SCENARIOS IN WHICH THE MOBILE PHONE RUNNING THE OBJECT TRACKING SOFTWARE MAY EXPERIENCE HAND JITTER FROM THE USER

Testing object tracking software with jitter is extremely crucial because in the real world, users may experience hand jitter while using tracking algorithms. The tracking efficiency of the algorithm is calculated with and without jitter.

The robotic arm can be programmed to exhibit jitter when moving the device under test while it is tracking the object under test. The impact of the induced jitter on the tracking efficiency can then be determined. The failures are triaged, and a pass/fail criterion is determined depending on the ability of the object tracking algorithm to handle jitter.

C. PROGRAMMING OF THE ROBOTIC ARM (USING ALL 6 JOINTS) TO POSITION THE DEVICE UNDER TEST RUNNING THE OBJECT TRACKING SOFTWARE FOR ZOOM-IN AND ZOOM-OUT EFFECTS

The robotic arm has been programmed to move closer to and farther away from the object being tracked. Moving closer to the test object will cause a zoom-in effect and moving away from the test object will cause a zoom-out effect. The robotic arm has been programmed to zoom in and zoom out while focusing on a PC monitor showing a video in which the object is also being tracked.

D. DEVELOPMENT OF POSTPROCESSING SOFTWARE FOR DETECTING THE BOUNDING BOX IN EVERY FRAME IN WHICH THE ALGORITHM TRACKS THE OBJECT IN ORDER TO DETERMINE THE TRACKING EFFICIENCY OF THE OBJECT TRACKING ALGORITHM

Postprocessing software has been developed to parse all frames tracked by the object tracking algorithm. A bounding box is drawn on the object if the object tracking algorithm can track the object. The Hough algorithm is used to detect the bounding box in these frames. The frame output (for every frame with a bounding box in which the object is tracked) is compared with the ground truth for that frame to calculate the number of frames in which tracking is successful.

Object tracking algorithms for use on mobile phones are becoming increasingly popular as the mobile revolution progresses. The performance of these algorithms is key to competitive differentiation among these embedded mobile devices. Thus, there is substantial focus on optimizing these computer vision algorithms with hardware-accelerated solutions to obtain a competitive advantage.

The main challenge is to validate the accuracy of the tracking algorithms on multiple hardware and software platforms. This paper reports the development of a reliable test setup for the automated testing of the ''touch-to-track'' feature. The development process required close collaboration with the software development teams to understand the software.

The testing requirements specified by customers and software development teams were carefully reviewed to ensure that appropriate test cases would be incorporated into the test plan. During the design and development of the test automation method for the validation of this computer vision algorithm, the testing points in the software architecture were reviewed. Methods of test automation were developed at two layers in the software: the Application Programming Interface (API) layer and the end-to-end Android layer. The test automation approach was designed after evaluating the requirements, and the final automation framework is able to satisfy the requirements for the execution of computer vision APIs at the API level and at the end-to-end Android level with a camera.

Computer-vision-based object tracking solutions that involve real-time tracking capabilities are critical for differentiating the feature sets of smartphones. The design and development of computer-vision-based software algorithms using embedded software are completely transforming the computational capabilities of smartphones, leading to a new era of improvement in user experience.

Consequently, enormous challenges are being faced in verifying the accuracy of these computer vision algorithms. Object tracking is an extremely common task, and this paper discusses test automation design and development for validating real-time object tracking algorithms.

A robotic arm has been programmed to automatically pick up a test phone and focus the test device on different moving test objects. Image processing algorithms such as Hough transformation are used to postprocess the frames in which each object is tracked and accurately determine whether the object was correctly detected and tracked in each frame. These automated tests provide the capability to determine the performance of an object tracker in terms of the efficiency of the object tracking algorithm.

The test results for object tracking efficiency that are obtained using the robotic arm setup with different panning angles with respect to the test objects are presented here. This test automation framework has been extensively used for commercialization efforts involving various software algorithms across multiple software products. The test automation algorithms for object tracking algorithms are discussed in detail in this paper.

This automated testing approach can be scaled to multiple software product lines. Any changes in a computer vision API that will impact functionality, stability and performance can be validated through the continuous integration tests and regression tests that are executed as part of the test automation framework.

The goal is to identify any newly emerging regression issues as early as possible in the software development cycle so that the code changes that cause these regression failures can, in turn, be fixed early in the software integration cycle.

Postprocessing of the test automation results and logs is critical for claiming complete automation without any human intervention. The postprocessing modules and software are designed to analyze every frame from the application under test and to analyze the output of the computer vision algorithm as run on the device under test. A postprocessing approach has been designed using MATLAB for the detection of the bounding boxes that are drawn by the tracking algorithm on all frames that contain the object being tracked. Then, algorithms can be applied to evaluate the correctness of the location of each bounding box by comparing it with the

ground truth to determine whether the bounding box contains the object in each frame.

These efforts have significantly reduced the amount of manual testing required and have effectively helped test automation teams to scale software to be deployed/executed across multiple product lines.

II. MOTIVATION

The primary motivation for the design and development of the touch-to-track test automation framework was to reduce the manual test overhead on test engineers and free up their time to work on new computer vision test software development. This also enables the investigation of the testing gates for the deployment of the test automation framework early in the software integration cycle to catch issues early. The requirements for the development of a reliable and robust test automation framework are listed as follows:

A. ENSURE THAT THE TOUCH-TO-TRACK APPLICATION DRAWS THE BOUNDING BOX IN EACH CAMERA FRAME IN WHICH THE OBJECT IS PRESENT AND COMPARE THE RESULT OF THE TRACKING ALGORITHM WITH THE GROUND TRUTH

The precision and recall are calculated to compare the object tracking result with the ground truth in each frame in which the object is tracked. The precision-recall results can be benchmarked across multiple software builds to ensure the quality of the algorithm.

B. DEVELOP AN ALGORITHM FOR PERFORMING IMAGE SEGMENTATION AND DETECTING THE BOUNDING BOXES GENERATED IN THE TOUCH-TO-TRACK APPLICATION

Once a binary image has been obtained, the vertical and horizontal lines defining the box location in the binary image must be identified. These horizontal and vertical lines are defined using kernels that respond to lines of single-pixel width with the specified orientation. The strategy for detecting the bounding box depends on finding the intersections between the perpendicular vertical and horizontal lines that have the same origin points. This algorithm enables the detection of the bounding box in every frame. Any frames in which no bounding box was detected can also be identified using this automated approach. The detected bounding box data are compared with the ground truth data to determine the efficiency of the object tracking algorithm in tracking objects frame by frame.

C. DETECT ANY JITTER PRESENT WHILE TRACKING OBJECTS IN THE TOUCH-TO-TRACK APPLICATION

Jitter may be observed during object tracking in each frame. Jitter may occur due to incorrect tracking of the object. It is necessary to develop robust postprocessing algorithms that can detect this jitter in the touch-to-track application.

D. DETERMINE WHETHER THE BOUNDING BOX OF THE OBJECT BEING TRACKED COVERS THE CORRECT OBJECT AND SURFACE AREA

If the bounding box does not accurately track the selected object, the root cause of the failure must be investigated. The test automation framework should be able to identify these issues.

E. EVALUATE THE PERFORMANCE OF THE TOUCH-TO-TRACK FEATURE IN BOTH CPU AND DSP MODES

Both CPU and DSP mode settings can be chosen for the touch-to-track application. Choosing the latter will lead to the selection of a code path that involves hardware acceleration using a DSP for tracking. The touch-to-track application is more efficient in terms of power consumption in DSP mode.

F. DETERMINE WHETHER HARDWARE ACCELERATION ENGINES ARE HELPFUL FOR DECREASING POWER CONSUMPTION AND IMPROVING KEY PERFORMANCE INDICATORS

Multimedia and computer vision algorithms are resource-intensive and require complex mathematical operations to be performed at a fast speed. Therefore, performing these calculations on a dedicated hardware processor such as a graphics processing unit or a DSP can lead to more optimized performance.

G. PERFORM A COMPETITIVE ANALYSIS WITH OTHER PHONE-BASED TRACKING ALGORITHMS

A competitive analysis of on-device tracking efficiency is performed with other algorithms that perform tracking on embedded devices.

III. BACKGROUND AND RELATED WORK

Template matching is an important task in computer vision. Dense local region detectors exist for extracting features suitable for image matching and object recognition tasks. Whereas traditional local interest operators rely on repeated structures that often cross object boundaries (e.g., corners and scale-space blobs), the sampling strategy adopted here is driven by segmentation and thus preserves object boundaries and shapes [1]. Dense matching methods typically enforce both appearance agreement between matching pixels and geometric smoothness between neighboring pixels [2].

Image matching can be achieved by means of pyramid matching. Fast kernel functions are available that map unordered feature sets to multiresolution histograms and compute weighted histogram intersection results in this space [3].

There exist several algorithms based on the extraction of features from a template and the subsequent matching of these extracted features to other parts of images. Such feature matching algorithms can leverage bottom-up segmentation. In contrast to conventional image-to-image

or region-to-region matching algorithms, there are methods that find corresponding points in an "asymmetric" manner, matching features within each region of a segmented image to features in a second, unsegmented image [4].

Methods also exist for efficiently comparing images based on their discrete distributions (bags) of distinctive local invariant features, without clustering descriptors [5].

Template matching can also be achieved by means of algorithms such as normalized cross-correlation and phase correlation. 2D normalized cross-correlation is performed by computing the normalized cross-correlation of matrices representing the template and the original image [6]. The resulting matrix will contain the correlation coefficients [7]. Phase correlation can similarly be used for pattern matching since it is an approach that can be used to estimate the relative translative offset between two similar images (digital image correlation) or other data sets. This approach is commonly used in image registration and relies on a frequency-domain representation of the data, which is usually calculated by means of fast Fourier transforms [8].

To test the computer vision features of object tracking algorithms, template matching algorithms are used in the framework described in this paper to detect the object being tracked in each frame. A 6-joint robotic arm is used for this purpose. The movements of the robotic arm must first be simulated using a simulator called WINCAPS III (DENSO Robotics). Then, the robot is programmed for automated testing. For the development of a control strategy for mobile robots, simulations are important for testing the software components, robot behavior and control algorithms in different surrounding environments. There exists a simulation environment for mobile robots based on Robot Operating System (ROS) and Gazebo [9]. Mobile robot simulations are a valuable tool for education, research and design. The last decade has seen a considerable increase in the development of new software tools for mobile robot simulations, which have reached different levels of maturity [10]. Using these robot simulators, it is possible to simulate the implementation of a particular robot control strategy within a short time. Simulators for a given robotic platform can be used to complete the grasp-and-place mission using a Gazebo virtual world and ROS [11]. Robots are extensively used for software testing. Robots are commonly associated with growth and the need for more mobile software testing [12]. The need for automation and high-quality test execution has increased with the explosion in the development of new feature sets for mobile software applications [13].

Robots are used in industry to facilitate the automation of repetitive tasks. Such robots can be efficiently used to pick up a device to be tested and then programmatically perform specific tasks in a repeatable manner. When failures are observed, it is possible to precisely determine the steps that were executed to produce the failure. This helps test engineers to precisely and accurately reproduce failures [14]. A robot setup can be efficiently used for testing because it can precisely repeat specified test sequences, and the test results can then be analyzed [15].

FIGURE 1. Image sequence for tracking in touch-to-track.

Robots can be controlled by programming their joints to execute specific test sequences [16]. For this purpose, it is necessary to design a trajectory plan for a robot based on its joints. Such trajectory planning can facilitate the execution of test scenarios that may involve tilting, rotation, etc., of the device under test [17]. Robot-based calibration is also crucial for ensuring that the movements of the robot are correct and precise [18]. To improve the quality and efficiency of software testing, methods of automated software testing have been widely used [19]. Test automation frameworks can be developed by using a robotic arm to incorporate more efficient tests and increase test coverage [20].

The challenges of robotic software testing extend beyond those of conventional software testing. Valid, realistic and interesting tests need to be generated for multiple concurrently running programs and hardware systems deployed in dynamic environments [21]. The execution of regression testing can benefit from the more accurate test results that can be obtained by using robot-based automation frameworks [22].

IV. TOUCH-TO-TRACK TEST AUTOMATION

Touch-to-track (T2T) is a feature that is provided as part of the Computer Vision software developed for the Android product lines.

A. OVERVIEW

Touch-to-track is an object tracker for arbitrary objects selected by the user. It is a real-time solution that starts tracking once the user draws a bounding box around an object. It can track multiple objects and recover tracking if a tracked object disappears and reappears in the scene. Touch-to-track works on the basis of tracking, detection, verification, and learning.

To test touch-to-track, a sequence of videos is played on a computer screen, and a mobile device is kept at a certain distance to maintain the screen within its FOV. The Android UI Automator tool [23] is used to draw the bounding box for testing, and a sequence of device screens is recorded

to test the performance of the tracker. The performance is measured in terms of bounding box detection performance in each frame and object detection performance within the bounding box.

B. TEST SETUP

The test laboratory contains the traditional one-device, one-monitor setup inside a single light booth. The light booth has six programmable and dimmable LED lights mounted on the top and sides. The purpose of the light booth is to create an environment in which the light intensity can be changed by an automation script while the device runs the object tracking application.

There is also a projector setup that projects moving images on a white wall in the laboratory for object tracking. The setup includes two projectors: one, a standard projector, projects background images to create different tracking environments, and the other, a rotational projector, projects multiple moving images to be tracked on top of the background. Therefore, the setup can be used to present scenarios with dynamically changing backgrounds and multiple object images to track.

Finally, the laboratory contains a 6-joint DENSO robotic arm that can be programmed to move the test device to different positions and perform various motions. In addition, by means of a remote-control toy car, the device can track a real 3D object from different angles while its moving. The robotic arm can also be used to simulate zoom-in, zoom-out and hand jitter scenarios. This test setup enables the automation of many device-motion-related tracking test cases.

C. TOUCH-TO-TRACK TESTING USING THE LIGHT BOOTH AUTOMATION WORKFLOW

A setup with a light booth and a PC monitor inside the booth has been designed. VLC media player is programmatically used to play videos. The mobile phone under test is held in a clamp and focused on the PC monitor.

Google's Android instrumentation class is used to launch the app. The necessary instrumentation code has been added to the Android application for the touch-to-track application. The addition of the instrumentation class allows automated tests to be triggered from the Android touch-to-track application and run-time arguments selected via the user interface to be passed to the application. These arguments include options for setting the mode of operation of the touch-to-track application, such as DSP mode with hardware acceleration and changing the preview resolution of the application.

The Android UI Automator is a tool that is used to draw the bounding box for tracking an object. Programmatically drawing the bounding box around the object in the camera preview of the touch-to-track application improves the accuracy of the bounding box. Please refer to Fig. 2 for the details of the touch-to-track test automation workflow. This workflow provides the details of the on-device test automation call flow.

As shown in the flowchart of the test automation workflow, the VLC player is used to render the videos on the PC monitor. The device under test is focused on the monitor. The test

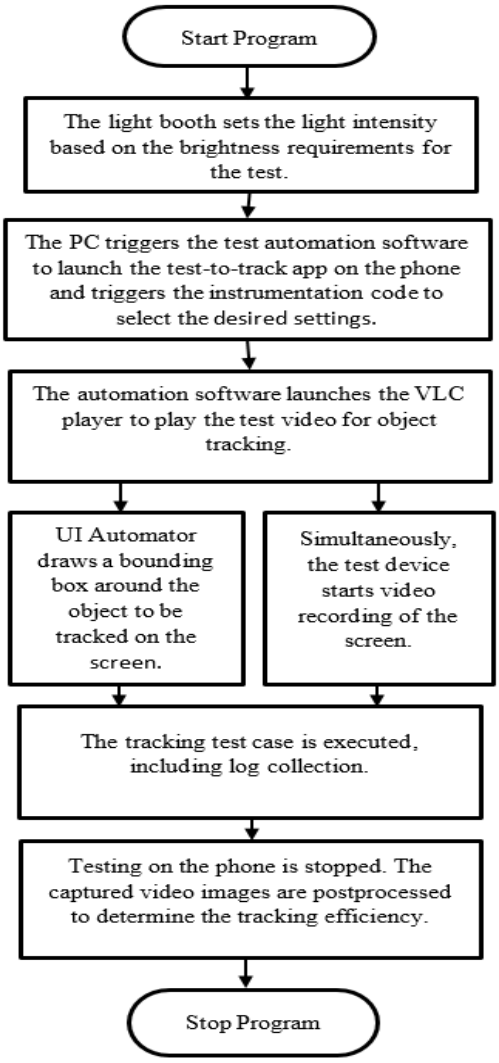

FIGURE 2. Flowchart of the touch-to-track test automation workflow.

automation software launches the VLC player to play the video on the monitor. The automation software also launches the touch-to-track Android application on the phone and draws the bounding box around the object to be tracked. The developed automation software records a video of the output frames that are rendered during touch-to-track execution. The automation software then extracts the video from the test device to the local machine and splits the video frames into static images. Subsequently, the presence of the bounding box is detected in the input frames using the bounding box detection algorithm explained in a later section to determine the tracking efficiency of the touch-to-track software.

D. AUTOMATED TOUCH-TO-TRACK TESTING WITH PROJECTORS

While testing touch-to-track using a display monitor playing videos is convenient, there are some downside of using this method. The display monitor will sometimes exhibit reflections or a Moiré effect that will interfere with the

FIGURE 3. Projected image tracking setup.

FIGURE 4. Projected image tracking against different backgrounds.

tracked image. More importantly, it is difficult to dynamically change the background environment when tracking an object based on video playback. Therefore, a method has been developed in which separate projectors are used to project the background images and the images to be tracked.

As demonstrated in Fig. 3, one standard projector projects different monocolor, blended color, and complex color background images. To present more challenging scenarios, it can project dynamically color-changing videos as well. The purpose of projecting these different background images is to simulate different tracking environments. This is helpful for studying how different colors, levels of complexity and motion characteristics of the environment affect the software's tracking ability and efficiency.

The other projector can be simply a holiday image projector. It projects only icon images without backgrounds. This projector is able to rotate internally to keep four or more images moving continuously in and out of view. The advantages of this projector are that it provides multiple moving images to track and offers different levels of motion speed. These projected images can be modified by changing the light filters. The automation process is identical to that for tracking with the display monitor. Depending on the test case requirements, the automation software will select single or multiple images to track, and the background can be selected from the available inventory.

Fig. 4 shows how the tracking setup environment looks during automated touch-to-track testing. In the depicted scenarios, the intent is to determine the tracking efficiencies achieved for the tracking of one to three images against no colored background and against monocolor, blended color and complex color backgrounds. These efficiency results can then be compared to simultaneously determine the effects of the tracking environment and the number of objects to be tracked.

E. TESTING TOUCH-TO-TRACK WITH A ROBOTIC ARM

Using the touch-to-track automation framework, many test cases have been automated by fixing the device under test in front of the monitor and playing different test videos for tracking. However, more test scenarios can be investigated if it is possible to move the test device while it is performing object tracking.

A solution has been found in which the test device is moved using a robotic arm. The robotic arm used for this purpose is manufactured by DENSO Robotics and has 6 joints, labeled J1 to J6, that rotate within given angular limits. The test device is held by a stepper motor at the tip of the robot arm. A display monitor is also set up on a test bench next to the robotic arm to display the test video.

One of the key use cases for testing touch-to-track is the execution of device zoom-in/zoom-out scenarios while tracking. This scenario applies, for example, when a drone is using active tracking to track a moving object. Both the object and the drone are moving, and the distance between them is constantly changing. Thus, the size of the tracked object on the video display also changes due to this zoom-in/zoom-out effect. To simulate this scenario during software testing, the robotic arm is used to move the test device closer to and farther away from the display monitor. Fig. 5 shows the workflow of this robotic arm automation program.

The WINCAPS III software, which provides a 3D simulation of the robotic arm, is used to program the arm. First, the arm angles J1 to J6 are adjusted such that the tip of the arm that is holding the test device is aligned facing the video display monitor. The test device is placed at a certain distance from the monitor as the initial starting position, as described in the flowchart. At this time, the automation software draws the bounding box around the object on the test device, and the touch-to-track software starts to track it.

To simulate the zoom-in scenario, the robotic arm is programmed to rotate each of the 6 arm joints to move the arm to a position where the tip of the arm is closer to the display, as shown in Fig. 6. In this way, a zoom-in effect is applied to the displayed object that is being tracked. If the touch-to-track software still recognizes the enlarged object image, object tracking should continue. Otherwise, if tracking is lost, the automated postprocessing algorithm will detect the images with missing bounding boxes and determine that the test case has failed. The same method is applied for the zoom-out scenario. In this case, the tip of the robotic arm is moved backward, farther away from the display than in the initial position, as shown in Fig. 6. In the zoom-out scenario, the object image appears smaller than the image

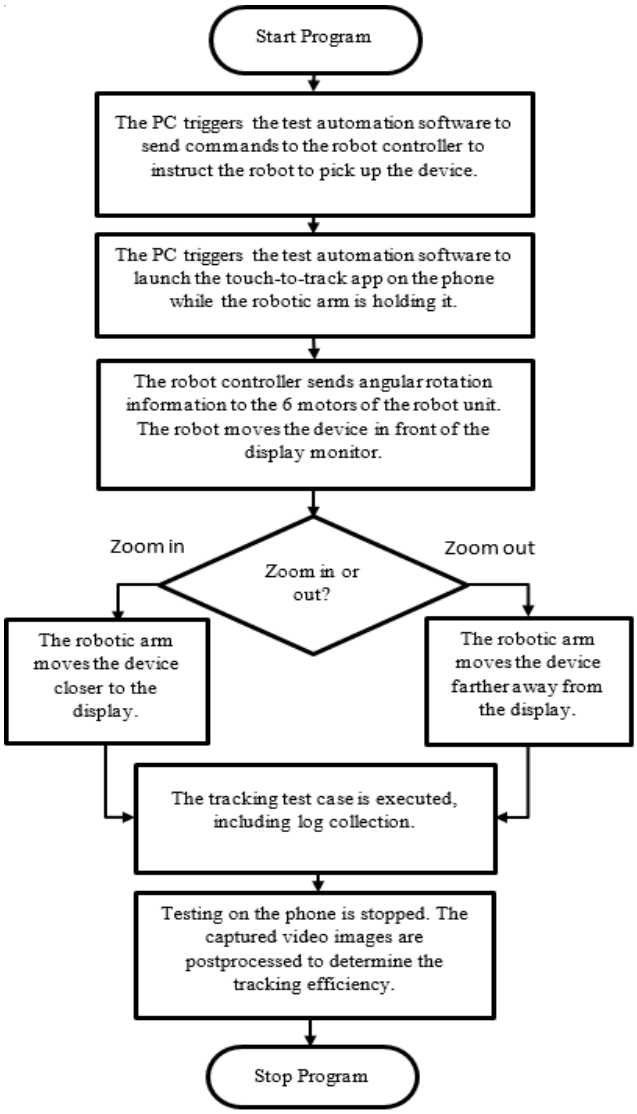

FIGURE 5. Flowchart of touch-to-track test automation using a robotic arm.

being tracked. This zoom-in/zoom-out test automation approach enables the validation of the software's object tracking performance when the shape of the object image being tracked remains the same but the size changes.

Fig. 7 shows images of the results of object tracking in automated tests of the zoom-in and zoom-out scenarios. In this case, the touch-to-track software is able to continuously track the moving object in the video in both scenarios.

Another, more challenging, use case that the robotic arm can simulate is angular motions of the device while focusing on tracking a real moving object.

First, the robotic arm is programmed to hold the test device at the base level where the test object is resting on a bench. In this case, the test object is a programmable remote-control toy car. In the touch-to-track software, the automation software draws a bounding box for tracking this toy car. Then, the robotic arm executes an angular motion to raise the test device to 30° above the toy car while still focusing on it and

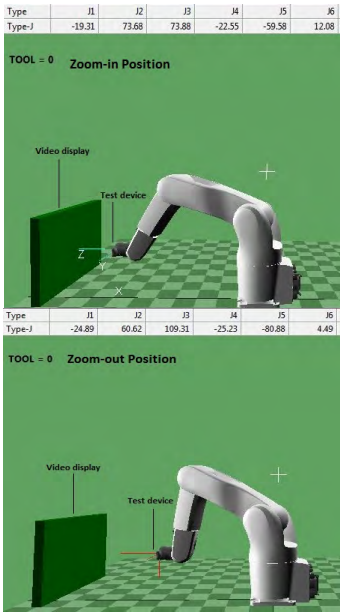

FIGURE 6. Robotic arm in zoom-in and zoom-out positions.

FIGURE 7. Zoom-in and zoom-out results.

maintaining the same distance. The toy car moves forward at a specified speed. The touch-to-track software should continue to track it while it remains in the device sensor's FOV. The entire tracking process is recorded by the device for the postprocessing of the tracking results. These processes are repeated for the 30° to 60° angular range and the 60° to 90° angular range, as illustrated in Fig. 8.

A way has been found to program the robotic arm such that the tip holding the device will form a specific angle with the object being tracked. In this case, the goal is to keep the remote-control car 100 cm away from the center base of the robot. In addition, the test device should maintain a constant distance of 60 cm from the car. To move the robotic arm 45°

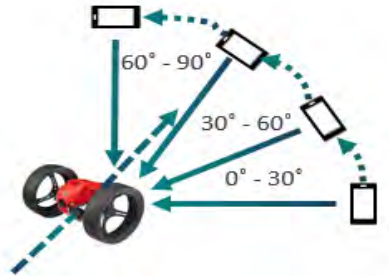

FIGURE 8. Tracking angles with respect to a toy car.

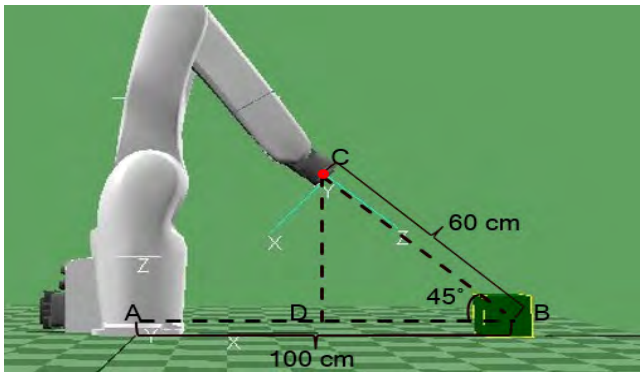

FIGURE 9. Diagram for calculating the coordinates of the robotic arm.

above the car, for example, the necessary (x, y, z) coordinates of the robotic arm's tip must be determined. As demonstrated in Fig. 9, the lengths of line segments AB and BC are known, and the value of the angle B is also known. It is necessary to find the length of line segment AD as the x-axis coordinate and the length of line segment CD as the z-axis coordinate; the y-axis coordinate is 0 since the robotic arm moves only in the xz plane.

Triangle BCD is a right triangle, so the length of CD can be calculated as follows.

$$CD = BC \times \sin 45^\circ = 60 \times \frac{1}{\sqrt{2}} \approx 42.4\ cm$$

Since the length of BD is equal to the length of CD, the length of AD can be determined as follows.

$$AD = AB - BD = AB - CD = 100 - 42.4 = 57.6\ cm$$

The (x, y, z) coordinates of the tip of the robotic arm are thus calculated to be (57.6, 0, 42.4), and the robotic arm moves only in the xz plane. The 6 joints of the robotic arm will be automatically positioned to move to these coordinates. Using this method, the robotic arm can also be programmed with additional angles for moving the test device with respect to the object being tracked.

This automated testing method poses a challenge to the touch-to-track algorithm. Since the test object is moving and the angle of the test device is changing, the shape of the image of the test object that is captured by the sensor could change substantially compared to the shape that was learned by the algorithm. Moreover, as the test object starts to accelerate, the touch-to-track software must be able to respond quickly and continuously track the fast-moving object from different angles. Please refer to Fig. 10 for snapshots of the test setup and the testing process.

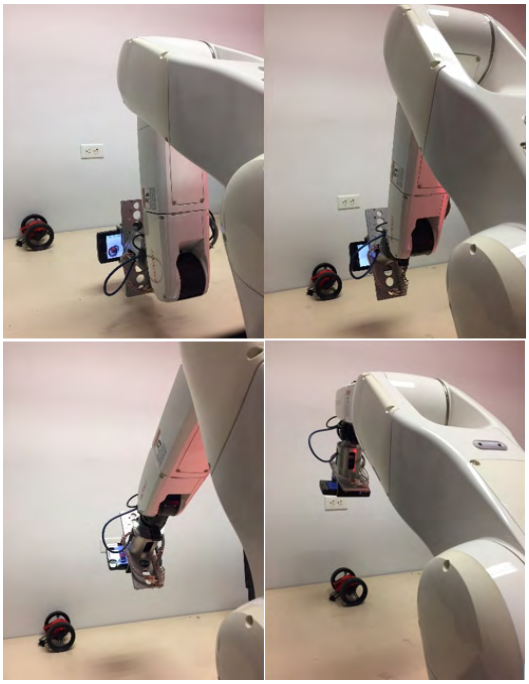

FIGURE 10. Test setup and snapshots of the testing process.

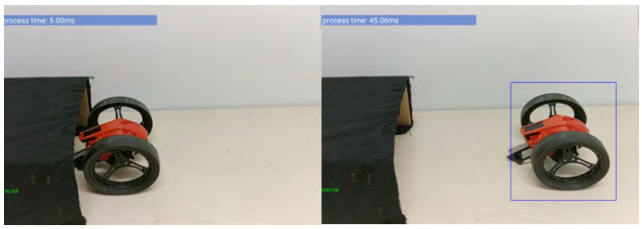

FIGURE 11. Video frames of the re-establishment of tracking after occlusion.

A few methods of comparing the tracking performance of different object tracking solutions using this automated testing system have been designed. One is to measure how quickly each object tracking solution can re-establish tracking after object occlusion. The software first establishes the tracking of the remote-control car at the base angle. The car moves through a tunnel at a specified speed and comes out the other end, as illustrated in Fig. 11. The software then re-establishes the tracking of the car. This test is repeated for different car speeds. The test is also repeated for each measuring angle interval. The resulting videos are postprocessed to count the number of frames elapsed between the frame in which the car has completely emerged from the tunnel to the frame in which the bounding box reappears. The number of frames required to re-establish tracking is then divided by the frame rate of the video in frames/sec to determine the time required to re-establish tracking for a performance comparison.

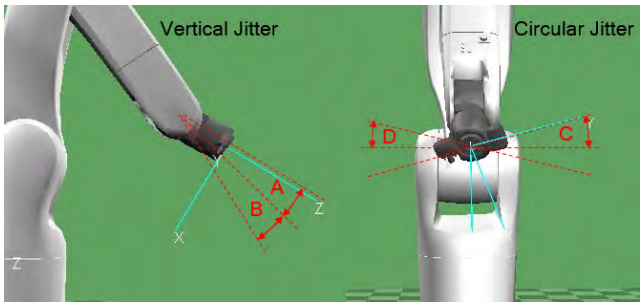

FIGURE 12. Diagrams of vertical and circular jitter motions.

A scenario with simulated hand jitter while tracking an object is another application of the developed test system. It is simply necessary to add simulated hand jitter on top of the angular motion program for the robotic arm. At the measuring angles in each angular range, two types of jitter motions can be added. One is vertical jitter, which is introduced by programming the robot to perform alternating motions of the same magnitude between angles A and B on joint 5, as shown in Fig. 12. The other is circular jitter, which is similarly introduced by moving joint 6 in an alternating manner between angles C and D.

Since the robotic arm offers a programmable motion speed option, the speed can be adjusted to vary the number of alternating motions per second to control the hand jitter frequency. Hand jitter during object tracking may cause the software to temporarily lose track of the object in some frames, especially at higher hand jitter frequencies. This test is designed to compare the object tracking efficiencies achieved at different hand jitter frequencies and different tracking angles.

F. BOUNDING BOX DETECTION

Bounding box detection is the first step in validating the tracking of the touch-to-track feature. By detecting the bounding box in each frame of a recoded screen sequence, it is possible to determine whether tracking is being performed in each frame. To detect the bounding box, each frame of the recorded screen sequence is treated as a static image for processing in MATLAB.

In the touch-to-track feature, the bounding boxes are color coded; for example, while tracking the first object, a blue box should appear. To identify a blue bounding box from an image (I), a binary image (Ibinary) should first be obtained.

$$Ibinary(x, y) = I(x, y, 1) == 0, \quad I(x, y, 2) == 0, \quad I(x, y, 3) == 1$$

Once the binary image has been obtained, the vertical and horizontal lines in the binary image must be identified to detect the box location. Horizontal (Ihorizontal) and vertical (Ivertical) lines are defined using kernels (Fx, Fy) [25] that respond to lines of single-pixel width with a specified orientation.

$$Ihorizontal = conv(Fx, Ibinary)$$
$$Ivertical = conv(Fy, Ibinary)$$

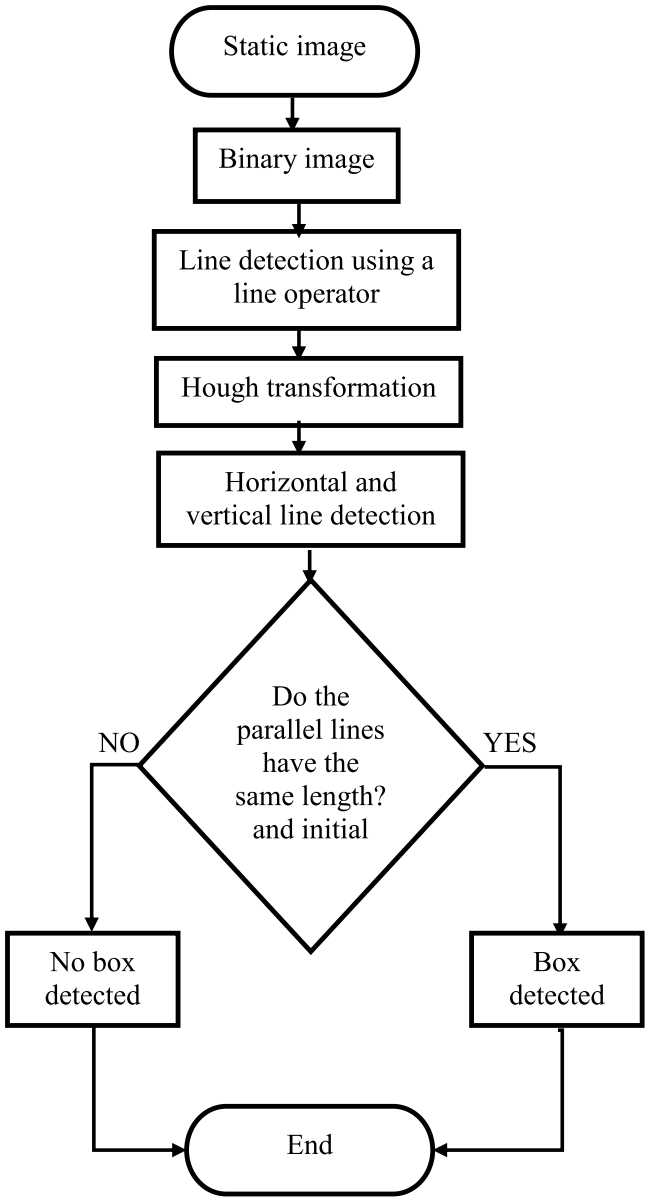

FIGURE 13. Flowchart of bounding box detection.

The kernels (Fx, Fy) are tuned to be sensitive to light lines against a dark background. To refine the search for the horizontal and vertical lines, the Hough transform is used to determine the orientations and lengths of the identified lines.

$$Lines_x, theta_x, rhox = Hough(Ihorizontal)$$
$$Lines_y, theta_y, rhoy = Hough(Ivertical)$$

Once the horizontal and vertical lines have been identified, lines of equal length in the horizontal and vertical directions are sought. It is verified that the starting and end points of a vertical line are the starting points of two equal-length horizontal lines. This verification yields the four points of the bounding box. If such points are identified, then a bounding box is present in the given image. This detected box can then be used for further object detection.

The histogram of oriented gradients (HOG) algorithm must be used for line detection and bounding box detection.

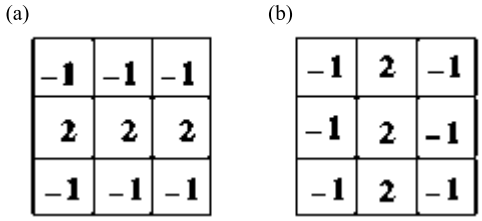

FIGURE 14. (a) The horizontal kernel (Fx) and (b) the vertical kernel (Fy), which respond maximally to horizontal and vertical single-pixel-wide lines, respectively.

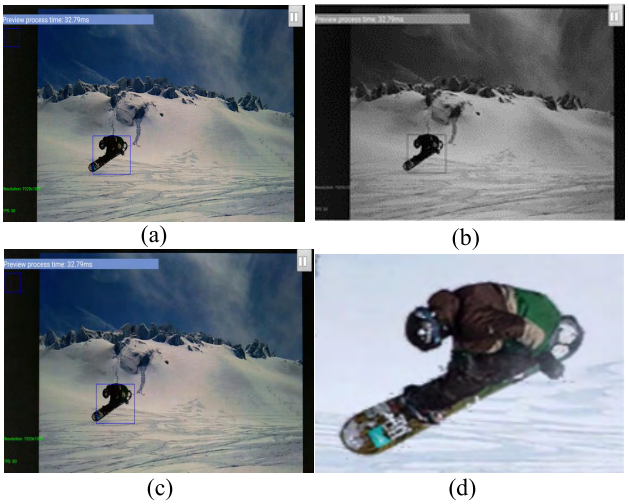

FIGURE 15. (a) Input image. (b) Binary image with bounding box detection. (c) Initial bounding box points in the input image. (d) Selected bounding box for object detection.

A Weiner filter can also be used to denoise the images to allow the HOG algorithm to function better. Moreover, the specific color (blue) of the object boundary box to be detected is known for every frame in the touch-to-track application. Therefore, a check is performed to ensure that the detected boundary box is blue in color. To ensure that the algorithm works under low-contrast conditions, the Weiner filter can be used for denoising. Image rectification techniques for sharpening the contrast, correcting the color, etc. can also be applied.

V. TOUCH-TO-TRACK APPLICATION POSTPROCESSING

A. TEMPLATE MATCHING USING SURF FEATURES

To measure tracking quality, some postprocessing must be included in the test automation framework. This postprocessing is performed with MATLAB. Postprocessing starts with the acquisition of screenshots from the device under test when the camera starts tracking the object shown in the video. Tracking images are obtained from the device in the form of screenshots.

In this paper, a postprocessing algorithm is proposed for quantifying tracking performance. The proposed algorithm (see Fig. 16 & 18) is designed to detect the bounding box, under the assumption that the object being tracked is inside the box. Object feature selection is performed for template matching in order to track the object in all screenshots.

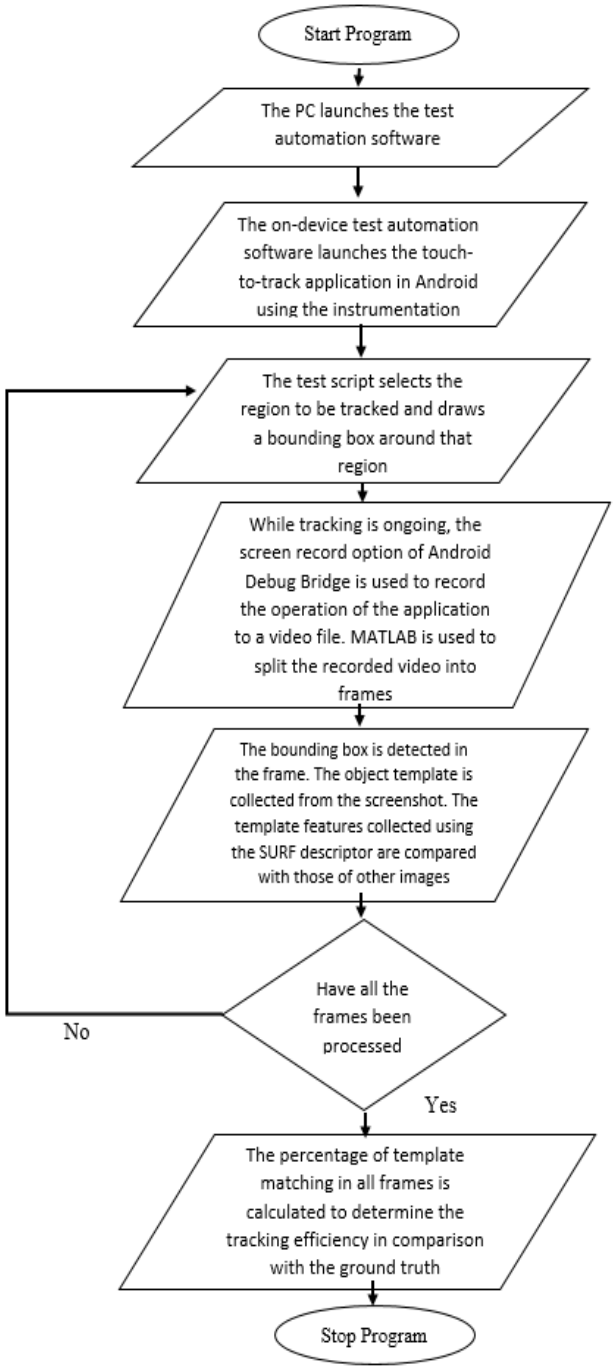

FIGURE 16. Flow chart of the touch-to-track postprocessing algorithm.

1) BOUNDING BOX DETECTION

The object being tracked is bounded by a rectangular box with RGB values of (0, 255, 0). Therefore, detection of this bounding box is required. The location of the box is determined by converting the image into a binary image by identifying the most connected green pixels.

2) OBJECT FEATURE SELECTION

The detected object inside the bounding box is used as a reference image for the rest of the screenshots. Object feature selection is required for further processing. The feature

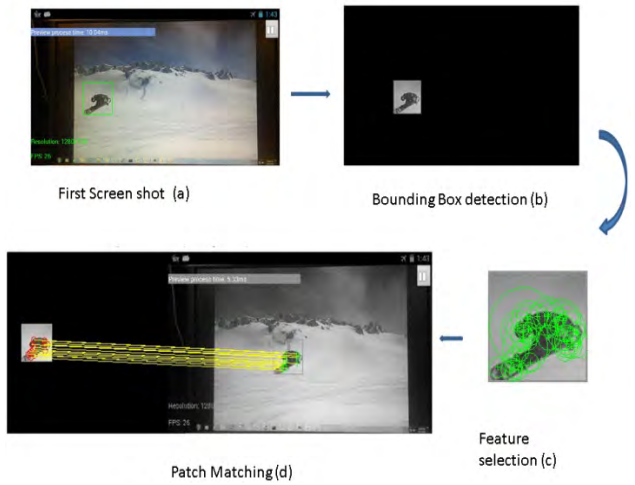

FIGURE 17. Stepwise presentation of the postprocessing algorithm. (a) First screenshot, used to learn the object inside the box. (b) Bounding box detected from the screenshot. (c) Collection of the features of the object. (d) Patch matching with the object in another screenshot to check the tracking.

selection process is performed using the SURF features of the object.

3) OBJECT TEMPLATE MATCHING

The SURF feature descriptor is used to obtain the features of the object template and the target image. The object is tracked by matching (using the matchfeatures function in MATLAB) the features of the object template and the target image. The direction of the object's motion is also calculated by finding the location of the object in each screenshot. The percentage of the object that lies inside the bounding box is also calculated.

B. TEMPLATE MATCHING USING NORMALIZED CROSS-CORRELATION

1) ALGORITHM DETAILS

The template is compared with the original image. The images are represented in matrix form using the imread function in MATLAB. The algorithm computes the normalized cross-correlation of the matrices representing the template and original images. The resulting matrix contains the correlation coefficients [24]. normxcorr2 uses the following general procedure [6], [7].

The cross-correlation is calculated in the spatial or frequency domain, depending on the image size. Calculate local sums by precomputing running sums.

The local sums are used to normalize the cross-correlation to obtain the correlation coefficients.

The implementation closely follows the formula from [24]:

$$\gamma = \frac{\sum_{x,y}[f(x,y) - \bar{f}_{u,v}][t(x-u,y-v) - \bar{t}]}{\left\{\sum_{x,y}[f(x,y) - \bar{y}_{u,v}]^2 \sum_{x,y}[t(x-u,y-v) - \bar{t}]^2\right\}^{0.5}}$$

where f is the image, $\bar{t}$ is the mean of the template, and $\bar{f}_{u,v}$ is the mean of f(x,y) in the region under the template.

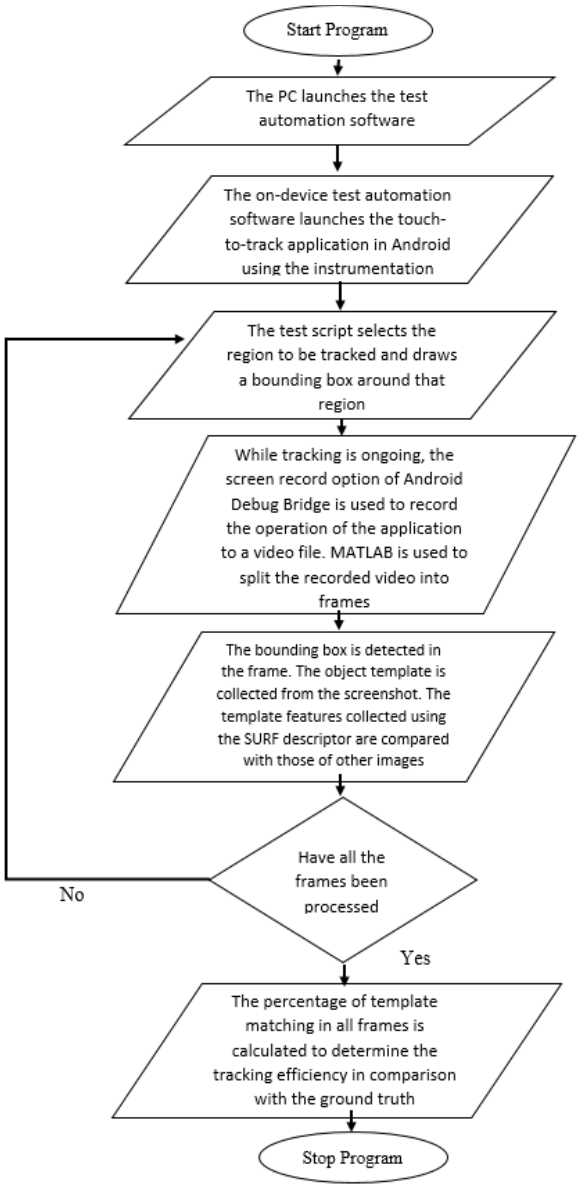

FIGURE 18. Test automation workflow for template matching using normalized cross-correlation for the object tracking efficiency calculation.

2) TEST AUTOMATION WORKFLOW WITH TEMPLATE MATCHING USING NORMALIZED CROSS-CORRELATION

Please refer to the flowchart (Fig. 18) for the details of the test automation workflow with the template matching algorithm using the normalized cross-correlations between the template (extracted from the bounding box) used for object tracking and the original images. The detection of a valid template match proves that the object represented by the template exists in the frame. The object detection result is compared against the ground truth to calculate the tracking efficiency.

C. TEMPLATE MATCHING USING PHASE CORRELATION

1) ALGORITHM DETAILS

Consider two input images ga and gb [8].

FIGURE 19. This figure shows the original image. The template is extracted from the bounding box. The template is compared with all subsequent images to determine whether the object is present in later frames. The results are compared with the ground truth.

FIGURE 20. The normalized cross-correlation is used to match the template with the images in which the object is tracked. The area identified through template matching is shown by the blue box. The algorithm exactly overlays the blue box on the bounding box, showing good accuracy of template matching.

A window function (e.g., a Hamming window) is applied to both images to reduce edge effects (this may be optional depending on the image characteristics). Then, the discrete 2D Fourier transforms of both images are calculated.

$$\mathbf{G}_a = \mathcal{F}\{g_a\}, \quad \mathbf{G}_b = \mathcal{F}\{g_b\}$$

The cross-power spectrum is calculated by taking the complex conjugate of the second result, multiplying the Fourier transforms together elementwise, and then normalizing the resulting product in an elementwise manner:

$$R = \frac{\mathbf{G}_a \odot \mathbf{G}_b^*}{|\mathbf{G}_a \odot \mathbf{G}_b^*|}$$

where ◯ is the Hadamard product (entry wise product) and the absolute values are also taken in an entry wise manner. The results are written out entry wise with element indices (j,k).

2) TEST AUTOMATION WORKFLOW WITH TEMPLATE MATCHING USING PHASE CORRELATION

Phase correlation is applied to match the template with the original image. In this way, it is possible to identify whether the object being tracked is present in the image. The total number of images containing the object is calculated, and the template matching output is compared with the ground truth for each frame. The details of the test automation workflow with template matching using phase correlation are provided in Fig. 22.

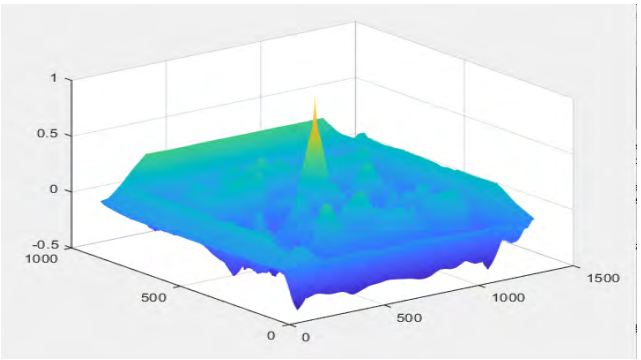

FIGURE 21. Cross-correlation results, displayed as a surface.

VI. RESULTS

A. TEST RESULTS WITH CHANGING BACKGROUNDS

Using the different test tools and setups introduced above, the touch-to-track software can be comprehensively tested. In touch-to-track testing with varying colored backgrounds, three background scenarios were tested: no colored background, a uniformly colored background and a blended color background. These tests were repeated for the tracking of one, two and three objects, each tracked for three minutes. The touch-to-track tracking processes were recorded by the device, and the resulting videos were collected for analysis. Fig. 25 shows sample result frames from these tests.

There were two purposes of this test. One was to compare the three template matching algorithms mentioned in the previous section and see which one worked the best in correctly detecting the bounding boxes and the content inside the bounding boxes. The other purpose was to determine the extent to which different types of backgrounds and numbers of images to be tracked affect the tracking efficiency.

To compare the accuracy of the three template matching algorithms, the same number of frames were collected from each of the scenario videos, such as those in Fig. 25. Then, postprocessing was run on these image frames using each of the three template matching algorithms to obtain the tracking efficiency results. After that, the videos frames were manually inspected by the tester to obtain the accurate tracking efficiencies, and they were used as the ground truth. The algorithm with the tracking efficiency results closest to the ground truth was the most accurate one. Table 1 shows the matching results obtained with the normalized cross-correlation, phase correlation and SURF-based feature extraction combined Matchfeatures template matching algorithms plus HOG bounding box detection, as well as the ground truth results from manual inspection.

From the results in this table, it appears that the HOG bounding box detection with SURF plus Matchfeatures

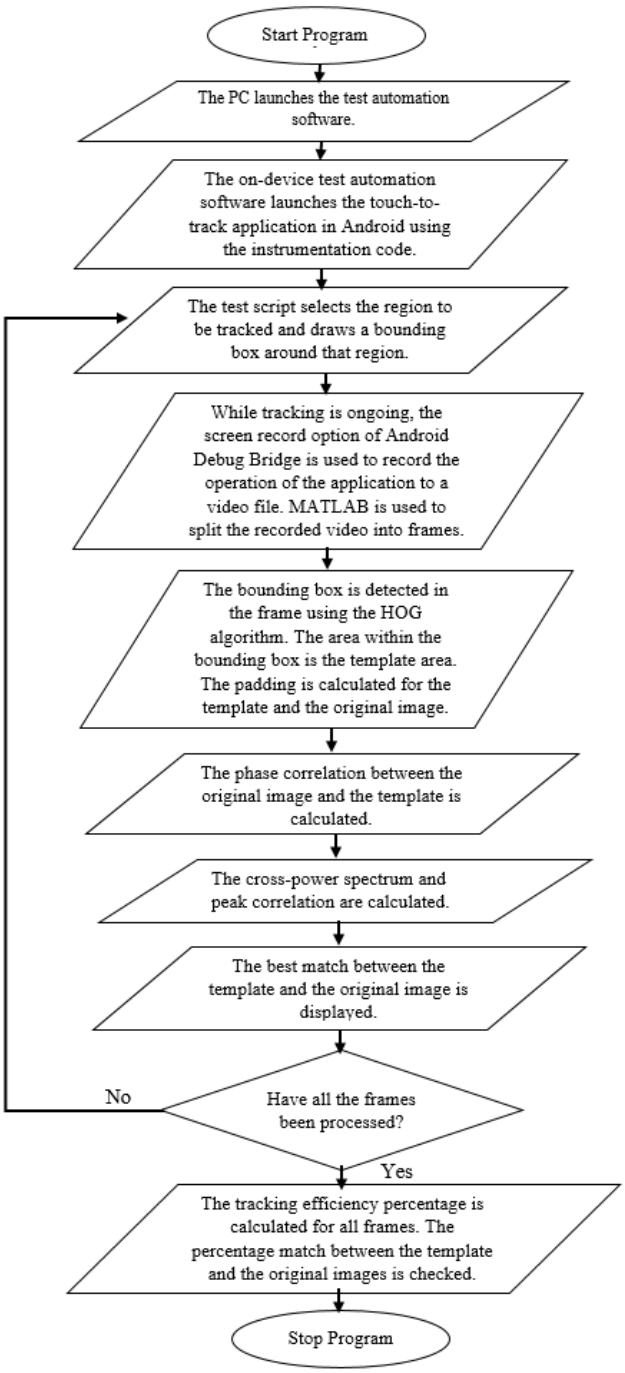

FIGURE 22. Test automation workflow for template matching using phase correlation for the object tracking efficiency calculation.

template matching algorithm had the closest matching percentages with the ground truth in all scenarios. Fig. 26 shows bar graphs breaking down the results for the three template matching algorithms comparing with the ground truth.

These graphs showed that the Normalized cross-correlation and the Phase correlation template matching algorithms were getting higher tracking efficiency results than the software's actual tracking efficiency. This indicated these two algorithms were detecting the bounding boxes but not effectively matching the content inside the boxes with

FIGURE 23. This figure shows the original image. The template is extracted from the bounding box. The template is compared with all subsequent images to determine whether the object is present in later frames. The results are compared with the ground truth.

FIGURE 24. The phase correlation is used to match the template with the images in which the object is tracked. The blue bounding box, representing the identified matching area, overlaps with the original bounding box (shown in black), demonstrating good accuracy of template matching.

FIGURE 25. Result frames from touch-to-track testing with varying colored backgrounds.

the templates. The bounding boxes could be still present and tracking content were already partially or completely disappeared from the video frames.

In addition, Fig. 27 shows another example from MATLAB that these two template matching algorithms resulted an incorrectly match. In the figure, the blue template matching box drawn on the Christmas tree corresponds to a correct match since the bounding box was drawn on the correct content. On the other hand, the matching box drawn next to the reindeer represents an incorrect match produced by the algorithm since the bounding box does not contain the correct content.

On the other hand, looking at the bar graph of the ground truth from Fig. 26, the tracking efficiency drops indicated that both the background color and the numbers of objects being tracked had effects on the touch-to-track software tracking efficiency. The tracking efficiency decreased as the background and environment color became more complex. Additionally, the tracking efficiency decreased as more objects

TABLE 1. Comparison of the accuracy results of different template matching algorithms with the ground truth.

Background Type	No Colored Background	Uniformly Colored Background	Blended Color Background
One-Object Tracking			
HOG + Normalized cross-correlation	100.0%	100.0%	99.5%
HOG + Phase correlation	96.3%	95.5%	94.6%
HOG + SURF + Matchfeatures	97.2%	94.0%	91.3%
Ground truth (manual inspection)	97.0%	93.6%	90.2%
Two-Object Tracking			
HOG + Normalized cross-correlation	96.5%	94.9%	92.8%
HOG + Phase correlation	95%	93.8%	92.6%
HOG + SURF + Matchfeatures	92.0%	89.2%	86.1%
Ground truth (manual inspection)	92.6%	88.6%	85.3%
Three-Object Tracking			
HOG + Normalized cross-correlation	92.0%	90.1%	90.5%
HOG + Phase correlation	91.3%	89.5%	90.0%
HOG + SURF + Matchfeatures	88.1%	86.6%	82.5%
Ground truth (manual inspection)	87.7%	85.5%	81.5%

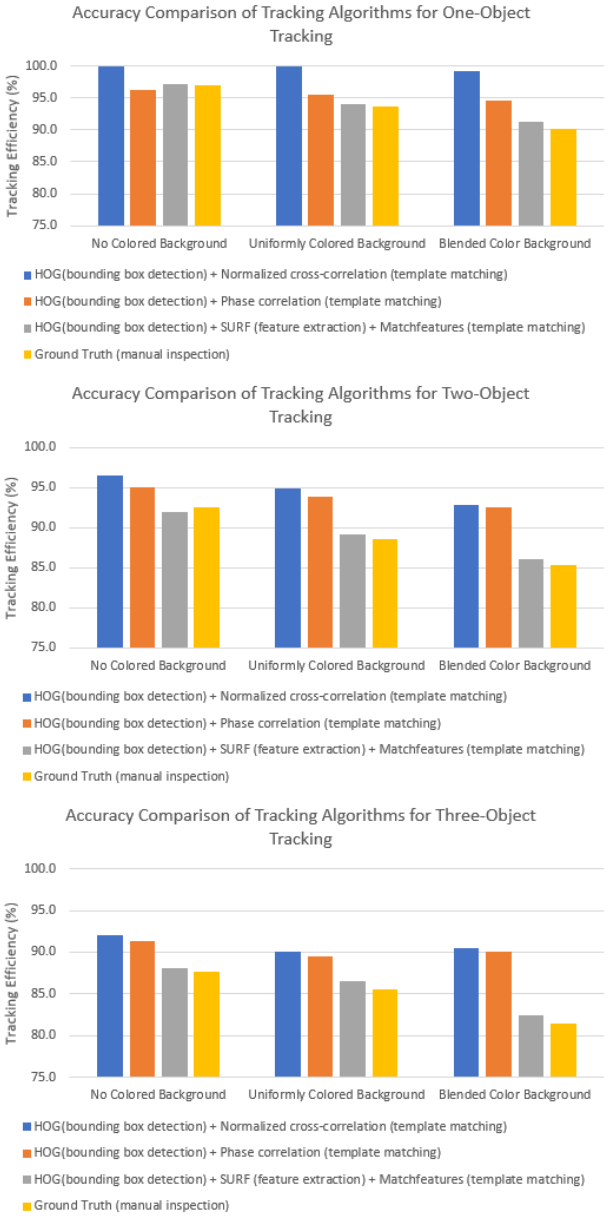

FIGURE 26. Graphs comparing the accuracy of the different template matching algorithms in the one-, two- and three-object tracking scenarios with the ground truth.

were tracked. And the touch-to-track software showed a steady tracking efficiency drop of 2% to 5% for both cases.

In this test, the tracking objects were moving in and out of the field of view continuously, which required time to re-establish tracking. The tracking efficiency drop with complexed background was likely due to the time it took to re-establish tracking on an object with colored and blended colored backgrounds was generally longer than the no colored scenario. Additionally, some objects were harder to track than others because their colors had less contrast with the background; consequently, more time was required to re-establish tracking for these objects which resulted in more frames were without bounding boxes for the colored background scenario. For the tracking efficiency drop with the number of objects factor, was likely because the increasing number of target-images resulted in a higher chance of image overlap. Overlapping object images are more likely to have different appearances than those of the templates, posing a challenge for both the tracking software and the postprocessing template matching algorithm, causing matching to fail.

Since the SURF feature extraction plus Matchfeatures template matching algorithm was the clear winner, this algorithm was chosen to be implemented in the automated postprocessing procedure to obtain the tracking efficiency results for the rest of the tests.

B. TEST RESULTS WITH THE ROBOTIC ARM

Another way of testing touch-to-track software is to use the robotic-arm-based test automation approach described in the previous section. By combining this approach with frame-by-frame bounding box detection plus normalized

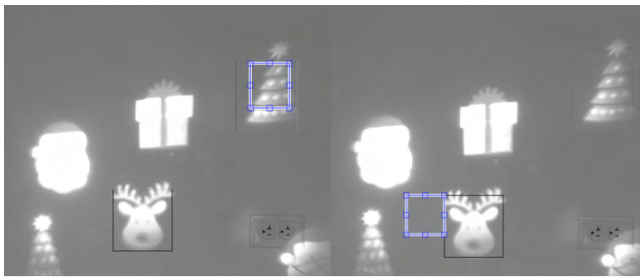

FIGURE 27. Sample images from MATLAB showing a correct matching case and an incorrect matching case.

TABLE 2. Results for the time required to re-establish tracking in object occlusion tests.

Comparison of Time to Re-establish Tracking				
Object Speed	**0.5 ft/s**	**1 ft/s**	**1.5 ft/s**	**2 ft/s**
15° Angle				
Touch-to-track time	0.29 s	0.38 s	0.47 s	0.65 s
3rd-party app time	0.77 s	0.81 s	0.84 s	0.89 s
45° Angle				
Touch-to-track time	0.35 s	0.43 s	0.54 s	0.71 s
3rd-party app time	0.80 s	0.84 s	0.89 s	0.95 s
75° Angle				
Touch-to-track time	0.47 s	0.51 s	0.64 s	0.79 s
3rd-party app time	0.85 s	0.89 s	0.97 s	1.09 s

cross-correlation template matching postprocessing, the touch-to-track software and a 3rd party object tracking solution were tested to compare their tracking performance.

In the object occlusion test, the object being tracked (in this case, the remote-control toy car) was registered at a 15° angle. It was set to run through a tunnel at different speeds of 0.5 ft/s, 1 ft/s, 1.5 ft/s and 2 ft/s. The time required to re-establish tracking of the car was measured 10 times at each speed. In addition, this test was repeated at device angles of 15°, 45° and 75° for both tracking solutions. The result videos for each condition were postprocessed, and the average times are listed in table 2.

The time required to re-establish tracking was measured by counting the number of frames from when the target object emerged from the occlusion to the frame in which a bounding box was drawn on the object and then dividing the result by the number of frames per second of the recorded video. The results in table 3 are also plotted in Fig. 28 to visualize the performance results for easier comparison.

A comparison of the overall results for the three different measuring angles reveals small average time increase of 0.077 seconds for the touch-to-track software and 0.061 seconds for the 3rd-party tracking software with a 30° increase in the measuring angle. This is due to the shape differences of the car being tracked as seen from different angles when it emerges from the other end of the tunnel.

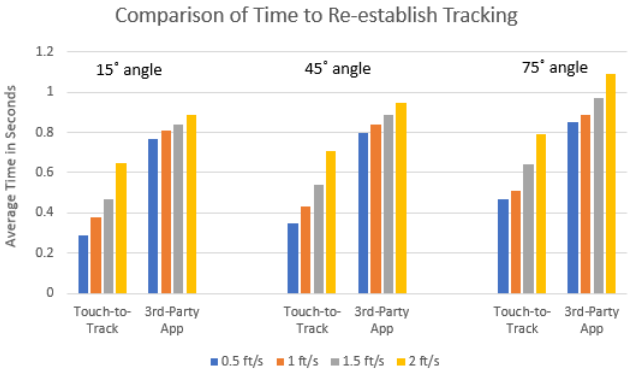

FIGURE 28. Comparison of the results for the time required to re-establish tracking.

TABLE 3. Comparison of tracking efficiency in the hand jitter scenario.

Comparison of Tracking Efficiency in the Hand Jitter Scenario				
Jitter Frequency	**No Jitter**	**1 Hz**	**2 Hz**	**3 Hz**
15° Angle				
Touch-to-track efficiency	99.3%	98.7%	96.2%	94.4%
3rd-party app efficiency	99.4%	97.8%	92.4%	87.7%
45° Angle				
Touch-to-track efficiency	98.6%	98.1%	94.7%	92.1%
3rd-party app efficiency	98.4%	97.2%	91.7%	85.9%
75° Angle				
Touch-to-track efficiency	98.1%	97.2%	92.6%	89.7%
3rd-party app efficiency	97.8%	96.1%	91.5%	84.8%

There is no significant difference between the two tracking solutions in this respect. However, when the times required to re-establish tracking with both solutions are compared for the same car speed, there are significant time differences between two solutions. Especially when the tracked object was moving at a low speed, the touch-to-track solution required only approximately half the time it needed by the 3rd-party tracking solution to re-establish tracking of the car after it emerged from the occlusion. At higher speeds, the tracking time of the touch-to-track solution increased more steeply, whereas the 3rd-party tracking solution showed a more consistent time increase. The faster the object was moving, the longer it took to re-establish tracking of the object.

Another tracking performance comparison between touch-to-track and the 3rd-party tracking solution was also performed in the hand jitter scenario. The remote-control car was moving at a constant speed, and the robotic arm moved the test device to 15°, 45° and 75° angles with respect to the car. The robotic arm simulated hand jitter at jitter frequencies of 1 Hz, 2 Hz and 3 Hz. Each test was again performed 10 times, and the test results obtained after postprocessing and averaging are listed in table 3.

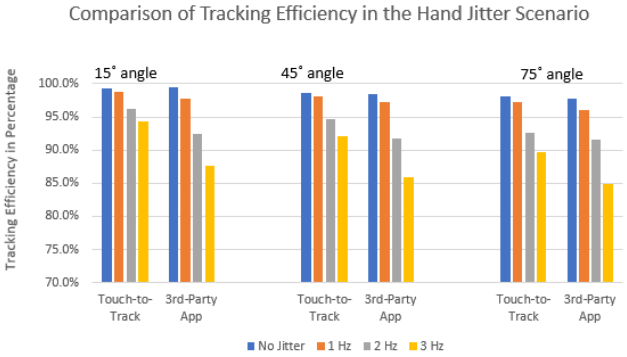

FIGURE 29. Comparison of tracking efficiency in the hand jitter scenario.

The tracking efficiency results were calculated by dividing the number of frames with bounding boxes by the total number of frames captured over the same length of time. To visualize and compare the results, the results in the table are plotted in Fig. 29.

By comparing the bar graphs for the three different angles, it can be seen that the angle had a small effect on the tracking efficiency for both tracking solutions. As the tracking angle increased, the tracking efficiency dropped slightly due to the differences in the appearance of the remote-control car from different angles. However, the hand jitter frequency was the major factor affecting the change in the tracking efficiency. When there was no hand jitter, both tracking solutions had similar tracking efficiencies. In addition, at the lowest hand jitter frequency of 1 Hz, both solutions had only a small drop in efficiency. At higher hand jitter frequencies, starting at 2 Hz, larger drops in efficiency were observed for both solutions. The hand jitter caused both solutions to have more difficulty keeping track of the object, and the faster motion speed of the device created more chances for the tracking mechanism to temporarily lose track of the object. However, from this hand jitter test, it was found that the touch-to-track solution had a smaller tracking efficiency drop than the 3rd-party tracking solution did, especially at a hand jitter frequency of 3 Hz. The overall results proved that the touch-to-track solution exhibited better tracking performance than that of the 3rd-party object tracking solution.

VII. CONCLUSIONS

The touch-to-track test automation framework described in this paper provides a platform for adding new test cases that are related to such factors as functionality, performance, and stability. In the emerging field of computer vision, it is highly important to understand the requirements of customers and work diligently to increase testing efficiencies to cater to the corresponding testing requirements.

Comparative tests were performed for the time required to re-establish tracking after object occlusion and for the tracking efficiency in the hand jitter scenario. It was observed that the object tracking solution with hardware acceleration performs better than the OpenCV-based solution with regard to the latency in re-establishing tracking and the robustness to hand jitter. The robotic arm was programmed for different angular movements and jitter scenarios and was used to test the tracking efficiencies of both object tracking solutions. The results proved that the touch-to-track solution using hardware acceleration performed better than the OpenCV-based solution.

This test automation framework has helped to scale test operations across multiple software products. A robotic arm laboratory has also been established for the execution of automated testing for specific test cases. Overall, the development of this touch-to-track test automation framework has yielded tremendous benefits for the commercialization of this feature in various software products.

The touch-to-track test automation framework provides comprehensive automated testing for the validation of a tracking algorithm that uses hardware acceleration. The robotic-arm-based test setup enables precise test execution. The robotic arm has been programmed to position the device under test with respect to a PC screen on which test-vector-based videos can be played. This will trigger the tracking algorithm to track the selected object.

The test automation framework includes image-processing-based algorithms developed to detect the bounding box that is used for tracking in each frame. Once the bounding box is detected the area under the bounding box is matched with the ground truth to determine if the bounding box is drawn over the correct object which is detected. This is important for determining whether the bounding box is correctly tracking the selected object.

The automation system needs to be able to pinpoint any performance degradation due to memory leaks, etc., while running stability test cases for object tracking. These enhancements of the test automation framework should allow regressions to be caught early and help to commercialize the object tracking solution on a shorter time scale.

REFERENCES

[1] J. Kim and K. Grauman, "Boundary preserving dense local regions," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 5, pp. 931–943, May 2015.

[2] J. Kim, C. Liu, F. Sha, and K. Grauman, "Deformable spatial pyramid matching for fast dense correspondences," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Portland, OR, USA, Jun. 2013, pp. 2307–2314.

[3] K. Grauman and T. Darrell, "The pyramid match kernel: Discriminative classification with sets of image features," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Beijing, China, Oct. 2005, pp. 1458–1465.

[4] J. Kim and K. Grauman, "Asymmetric region-to-image matching for comparing images with generic object categories," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, San Francisco, CA, USA, Jun. 2010, pp. 2344–2351.

[5] K. Grauman and T. Darrell, "Efficient image matching with distributions of local invariant features," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, San Diego, CA, USA, Jun. 2005, pp. 627–634.

[6] J. P. Lewis, "Fast template matching," in *Vision Interface*, vol. 95. Quebec City, QC, Canada: Canadian Image Processing and Pattern Recognition Society, 1995, pp. 120–123.

[7] R. M. Haralick and L. G. Shapiro, *Computer and Robot Vision*, vol. 2. Reading, MA, USA: Addison-Wesley, 1992, pp. 316–317.

[8] *Phase Correlation*. Accessed: Sep. 11, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Phase_correlation

[9] K. Takaya, T. Asai, V. Kroumov, and F. Smarandache, "Simulation environment for mobile robots testing using ROS and gazebo," in *Proc. 20th Int. Conf. Syst. Theory, Control Comput.*, Dec. 2016, pp. 96–101.
[10] P. Castillo-Pizarro, T. V. Arredondo, and M. Torres-Torriti, "Introductory survey to open-source mobile robot simulation software," in *Proc. Latin Amer. Robot. Symp. Intell. Robot. Meeting (LARS)*, Oct. 2010, pp. 150–155.
[11] W. Qian *et al.*, "Manipulation task simulation using ROS and gazebo," in *Proc. IEEE Int. Conf. Robot. Biomimetics*, Dec. 2014, pp. 2594–2598.
[12] 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. 2017.
[13] K. Mao, M. Harman, and Y. Jia, "Sapienz: Multi-objective automated testing for Android applications," in *Proc. 25th Int. Symp. Softw. Test. Anal. (ISSTA)*, Saarbrücken, Germany, Jul. 2016, pp. 94–105.
[14] D. Banerjee, K. Yu, and G. Aggarwal, "Robotic ARM based 3D reconstruction test automation," *IEEE Access*, vol. 6, pp. 7206–7213, Jan. 2018.
[15] M. Jasiński, J. Mączak, P. Szulim, and S. Radkowski, "Autonomous agricultural robot—Testing of the vision system for plants/weed classification," in *Automation 2018* (Advances in Intelligent Systems and Computing), vol. 743, R. Szewczyk, C. Zieliński, and M. Kaliczyńska, Eds. Cham, Switzerland: Springer, Mar. 2018, pp. 473–482.
[16] D. Banerjee, K. Yu, and G. Aggarwal, "Hand jitter reduction algorithm software test automation using robotic ARM," *IEEE Access*, vol. 6, pp. 23582–23590, Apr. 2018.
[17] D. Banerjee, K. Yu, and G. Aggarwal, "Image rectification software test automation using a robotic ARM," *IEEE Access*, vol. 6, pp. 34075–34085, Jun. 2018.
[18] 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.
[19] 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.
[20] D. Banerjee and K. Yu, "Robotic ARM-based face recognition software test automation," *IEEE Access*, vol. 6, pp. 37858–37868, Jul. 2018.
[21] 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.
[22] 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. 1–5.
[23] *UI Automator*. Accessed: Sep. 11, 2018. [Online]. Available: https://developer.android.com/training/testing/ui-automator.html
[24] *Normalized 2-D Cross-Correlation*. Accessed: Sep. 11, 2018. [Online]. Available: https://www.mathworks.com/help/images/ref/normxcorr2.html
[25] R. Milanese, S. Gil, and T. Pun, "Attentive mechanisms for dynamic and static scene analysis," *Opt. Eng.*, vol. 34, no. 8, pp. 2428–2434, Aug. 1995.

DEBDEEP BANERJEE received the master's degree in electrical engineering from the Illinois Institute of Technology. He has over10 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 the Software/Systems Development Engineer serving as the Test Lead for the Computer Vision Project. He is responsible for 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 currently involved in managing and developing software for the robotic arm used in the Computer Vision Laboratory.

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

GARIMA AGGARWAL is currently a Test Engineer with Qualcomm Technologies, Inc., USA, where he has actively involved in MATLAB postprocessing modules for computer vision features and various other automation projects.