IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 10, 2018, accepted December 5, 2018, date of publication December 11, 2018,
date of current version January 7, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2886272

Integrated Test Automation for Evaluating a Motion-Based Image Capture System Using a Robotic Arm

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

ABSTRACT The problem discussed in this paper is how to create an integrated test automation setup, in which the test engineer does not have to use a pendant to communicate with the robotic arm and a separate test script to trigger on-device test automation for testing computer vision algorithms, such as motion-based image capture. There is no communication between the pendant to control the robotic arm and the test script that launches the on-device test automation for the motion-based image capture system executing on a mobile phone. Therefore, the test engineer needs to manually trigger the programs for the robot pendant to start the robotic arm, the automation, and launch times to keep them in sync. The solution to this problem is to design a test automation program using a middleware software (such as ORiN2 middleware software provided by Denso Robotics) to bypass the dependency on the robotic pendant, which must be manually started. The programmatic usage of the ORiN2 software also helps to establish a communication between the lab computer and a robot controller. The controller access object is used by the ORiN2 middleware software to communicate with the robotic controller of a Denso robot. It directly controls the robotic arm motion by accessing the programming variables. The image capture accuracy is calculated based on the number of snapshots and the similarity of the snapshots taken if the mobile phone under test is stationary and a motion is injected in the test subject. We compared the test results of the hardware-accelerated solution with a third-party solution, and found that the hardware accelerated solution is 12% more accurate for image capture accuracy at a medium panning speed. This paper discusses the integrated test automation framework for testing motion-based image capture system using a robotic arm.

INDEX TERMS Robotics and automation, robots, computer vision, image processing.

I. INTRODUCTION

The problem statements described in the papers includes the manual control of the pendant to control the motion of the robotic arm. The second problem includes the non-communication between the test script starting the on-device tests for the motion-based image capture system on a mobile phone and the robotic arm used for executing the tests.

The usage of the ORiN2 middleware has helped us to eliminate the need of using a pendant to control the robotic arm's motions and directly control the motions of the robotic arm by using computer-based automation code.

This technical paper proposes an integrated test automation setup to eliminate the manual trigger of test automation of the on-device and robotic arm test systems. The deployment of this integrated test automation has helped us to reduce manual testing and scale test efficiencies across multiple software product lines.

The distinctive features discussed in this paper are:

1. INTEGRATED TEST AUTOMATION FRAMEWORK FOR EXECUTING AUTOMATED TESTS FOR EVALUATING THE FUNCTION, PERFORMANCE, ETC. OF THE MOTION-BASED IMAGE CAPTURE ALGORITHMS IMPLEMENTED IN MOBILE PHONES USING A ROBOTIC ARM.

We have developed a test automation framework which automates the tasks of triggering the robotic arm programs using ORiN2 middleware to directly communicate between a PC and a robotic controller. The test automation also supports the on-device test automation launch and synchronization between the robotic arm motions and the on-device test automation tasks. This test automation framework has

allowed us to expand test automation and to add relevant test scenarios to evaluate more test cases.

2. ELIMINATION OF THE MANUAL USAGE OF THE ROBOTIC PENDANT AND MANUAL TRIGGER OF AN ON-DEVICE TEST AUTOMATION SCRIPT TO SYNCHRONIZE THE TIMINGS OF THE TEST OPERATIONS WHILE BOTH PROGRAMS ARE RUNNING SIMULTANEOUSLY.

The problem with the robotic pendant is that the test engineer must program the test automation in the pendant and manually launch the program. Additionally, the test engineer must start the on-device test automation when the robotic arm picks up the mobile undertest and is close to the test subject to launch the camera preview of the motion-based image capture systems. Therefore, the synchronization between controlling the pendant and launching the test automation is manual and causes errors, etc. Elimination of the robotic pendant by allowing the PC to directly communicate with the robot controller for controlling the robotic arm and control the on-device test automation solves the problem of the manual intervention.

3. COMPILING A VISUAL BASIC 6-BASED EXECUTABLE TO DIRECTLY USE ORIN2 MIDDLEWARE TO COMMUNICATE WITH THE ROBOTIC CONTROLLER (RC7) OF THE ROBOTIC ARM.

We bypassed any GUI-based applications to control the robotic controller by compiling Visual Basic code which uses controller access objects. The controller access objects are used by the ORiN2 middleware program to directly communicate with the robotic controller of the robot. This way we directly communicate with the robotic controller from the PC and control the robotic arm movements.

4. ON-DEVICE TEST AUTOMATION FOR EVALUATING THE PERFORMANCE OF THE MOTION-BASED IMAGE CAPTURE SYSTEM SUCH AS LATENCIES, THE ACCURACY OF CAPTURE, ETC.

The on-device test automaton has been developed to accurately cause the motions of the test objects and to control the angular movement of the device under test with the robotic arm. The motion vectors of the frames are analyzed, and the reference frames are saved in the device. Therefore, the test setup is the same, and the same type of motion is injected in the test subject, so we can generate the same I frame. This way we can repeat the tests and check for the I frame consistency by the same motion injection in the test subject.

5. COMPETITIVE ANALYSIS OF THE COMPARISON OF THE PERFORMANCE OF THE HARDWARE-ACCELERATED MOTION-BASED IMAGE CAPTURE WITH 3RD-PARTY SOLUTIONS.

We compared the test results of the performance test scenarios of the hardware-accelerated solution with 3rd-party solutions, and we observed better performance metrics for the motion-based image capture algorithms running with hardware-acceleration. The hardware accelerated solution is implemented using digital signal processors which are used for evaluation of the motion estimation in the images.

The detailed results with a comparison are provided in the ''Results'' section of this paper.

II. MOTIVATION

The motivation points for this technical paper are provided as follows:

1. REDUCE MANUAL TESTING FOR TESTING COMPUTER VISION-BASED FEATURES SUCH AS MOTION-BASED IMAGE CAPTURE WITH A ROBOTIC ARM.

We need to eliminate the manual testing and strive towards deploying test automation as manual testing is tedious and error-prone. Since we had to trigger test automation manually and synchronize the timings of the test automation execution for the robotic arm and in device testing, it was time-consuming and involved the test engineer's time. With complete end-to-end automation deployment, test engineers can focus on test feature test planning and writing automation code for automating the new features.

2. INTEGRATION OF PC-BASED TEST AUTOMATION TO SIMULTANEOUSLY CONTROL MULTIPLE TEST SCRIPTS RELATED TO ON-DEVICE TEST AUTOMATION AND A ROBOTIC CONTROLLER CONTROL FOR A ROBOTIC ARM.

Without integration and communication between the test automation script, the PC cannot communicate with the code controlling the robotic arm. For example, if the on-device test automation code launches the on-device application for motion-based image capture and the camera preview does not display, the test execution may terminate as it may not make sense to proceed with the robotic arm-based test. Therefore, it triggers an event to the robotic arm, and it can terminate the panning operation. This way we can efficiently use the robotic arm and the on-device test automation times and save unnecessary time usage in the case of fundamental failures.

3. DEVELOP RELIABLE TEST AUTOMATION TO EVALUATE THE PERFORMANCE OF THE COMPUTER VISION USE CASE TESTING OF MOTION-BASED IMAGE CAPTURE.

The motion-based capture needs a reliable test setup based on the robotic arm which can inject controlled motions to the device and the subject under test. The robotic arm can be programmed to record the motion changes in the test subject or make motions relative to the speed of the test subject motions, and accuracy calculations or performance test cases can be performed.

4. AUTOMATE THE PERFORMANCE-BASED ACCURACY AND LATENCY TEST SCENARIOS USING A ROBOTIC ARM TO PROVIDE THE EXACT MOTIONS TO THE MOTION-BASED IMAGE CAPTURE APPLICATION TO EXTRACT I FRAMES AND BENCHMARK THE RESULTS WITH ACCURATE MOTION INJECTION.

The motion-based image capture involves checking for the changes in the motion vectors in the image capture frames and extracting reference frames based on changes detected in

the macroblock size-based motion vector changes per frame which can indicate changes in motion, etc. With the same test setup and controlled motion injections to the test subject, etc., the same I frame can be extracted and these can be benchmarked per the software build to verify the algorithmic integrity of the test automation system.

III. BACKGROUND AND RELATED WORK

There exist software frameworks for testing robot control systems in simulation [1]. Robotic operating systems (ROS) have been used extensively for simulating the robotic controls and movements so that it can be thoroughly tested before the programs are deployed in real-world robots [2]. Robotic operating systems have helped engineers to evaluate the control logic and check for its locomotion systems [3]. This helps to test robots before deploying them in action in automated manufacturing lines, etc. The robots can remote control the operation of the robot and guide its motions, etc., [4]. The robotic operations can remote human-in-the-loop grasp execution [5]. The software simulation of a robot control systems is crucial for successful tests and commercialization efforts in deployments for achieving automation.

Robots have been extensively used for software testing [6]. Since software testing involves the repetition of steps, it is convenient to program a robot to allow it to repeat the same steps. Multiple software frameworks exist today to test mobile phone applications with specialization to maximize test coverage and uncover software issues [7]–[8]. Robots can be used extensively to perform testing on mobile applications, which includes face recognition algorithms [9]. Camera-based applications can be tested using programmable robots to validate hand jitter reduction algorithms to achieve a better user experience [10]. Research has been conducted to test the software programs controlling multiple robots [11]. Mobile applications such as object tracking and image rectification have been automated by robots which are programmable and can test the mobile software with great precision [12], [13]. Robots can be manually controlled using interfaces that can be used to control the robot movements [14]. Robots can thereby be controlled and programmed to control its angular movements to perform application-specific test execution [15]. Robot-based tracking control design has been developed for high precision robot tracking [16].

Application-based software testing has thereby greatly benefited from using robots for testing software, and it has provided high precision test results due to its ability to repeat a test sequence accurately.

IV. SOLUTION

A. INTEGRATED TEST FRAMEWORK OVERVIEW

The robotic arm system is today's popular tool for end-to-end software app testing that involves motions. We developed multiple test programs and methods for computer vision-based apps using the robotic arm system that saved countless hours from manual testing and greatly improved the test accuracy. However, there were still some manual steps involved in the testing process. For example, the tester had to use a teach pendant connected to the robot controller to start and stop a robotic program saved on the controller. The controller sent commands to the robotic arm for executing the program. Additionally, the test script, using an Android debug bridge to control the software testing on the device, was triggered manually from a lab computer. The biggest problem here was that since the device test program and the robotic arm program were triggered parallelly, there was no communication between these two systems. The tester had to trigger them precisely with the timing to keep them in sync as demonstrated in Figure 1. To solve this problem and eliminate the manual work completely from this testing process, we adapted ORiN2 middleware to establish communication between the lab computer and the RC7 controller, enabling robotic arm control completely bypassing the teach pendant.

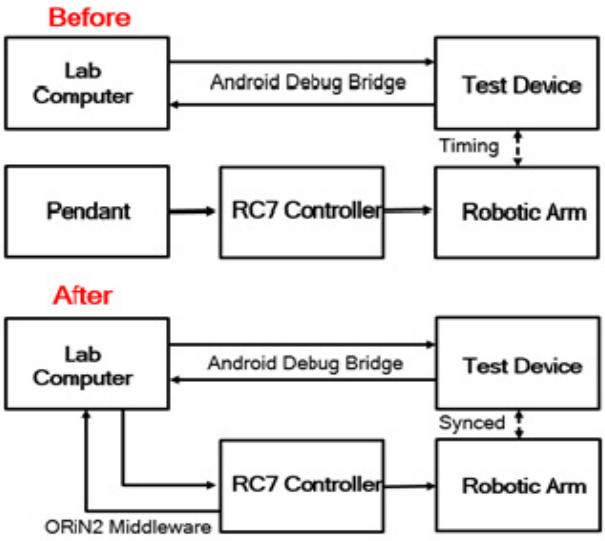

FIGURE 1. Test framework before and after using ORiN2 middleware for communication.

This improvement in our robotic arm testing system not only eliminates the manual triggering process on the robotic arm program but also enables the lab computer to receive feedback from the robot controller. This provides the ability for the test script to perfectly sync the timing between the testing on the device and the motion of the robotic arm.

B. TEST EXECUTABLE TO ROBOTIC ARM COMMUNICATION BLOCK DIAGRAM DETAIL

We program the robotic arm using the WINCAPS 3 software system due to its offline robot programming and simulation feature. The ORiN2 middleware is part of the WINCAP 3 package for factory level automation. It allows the PC to directly control the robot from a network via independent interfaces. The ORiN2 middleware supports multiple common programming languages. To set up ORiN2

middleware on the robotic arm system, first, we must install the ORiN2 framework dependencies on the controller, which includes the RobSlave.pac program that takes tasks from the ORiN2. Then, a network channel through ethernet connection needs to be set up between the lab PC and the RC7M controller with read/write permissions. After setting up ORiN2 on the system, our goal is to create an executable file compiled from the Visual Basic 6 programming language that uses ORiN2 middleware to communicate with the robot controller. Since the robot controller is an RC7 model, the CAO (controller access objects) provider is called NetwoRC. ORiN2 middleware uses the CAO engine through Rob Talk to communicate with RobSlave.pac program running on the controller. It can directly control the robotic arm motion by accessing the programming variables or calling another PAC program stored on the controller to run the robotic arm motion sequence. The block diagram is shown in Figure 2.

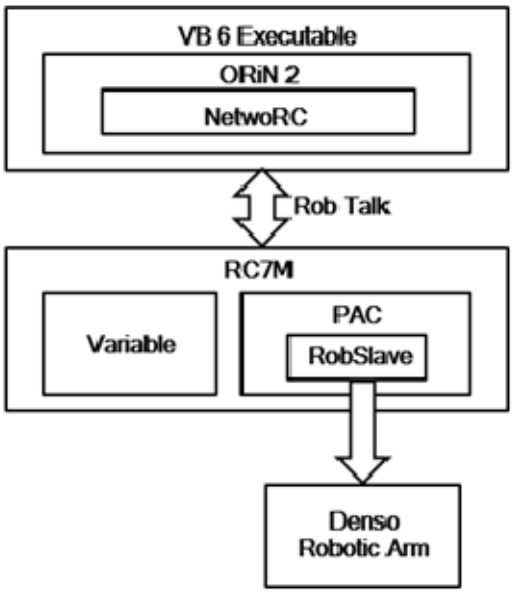

FIGURE 2. Block diagram for ORiN2 middleware communication with the Denso robotic arm.

C. THE VB 6 TEST EXECUTABLE WORKFLOW

Since the VB 6 executable enables communication between the computer and the robotic arm, it needs to be designed to take any robotic arm program input from WINCAPS 3 and run it on the robotic arm when it is executed. The detailed workflow of the VB 6 executable is shown in Figure 3 below. The precondition for running the executable is that the controller has to be set to external automatic mode, the motor has to be turned on, and the RobSlave.pac program has to be running on the controller. These steps are one-time efforts as part of the robotic arm system startup procedures. The executable will initialize the caoEngine and attempt to connect to the RC7 controller by adding caoWorkspace and caoController through providing the computer's ethernet

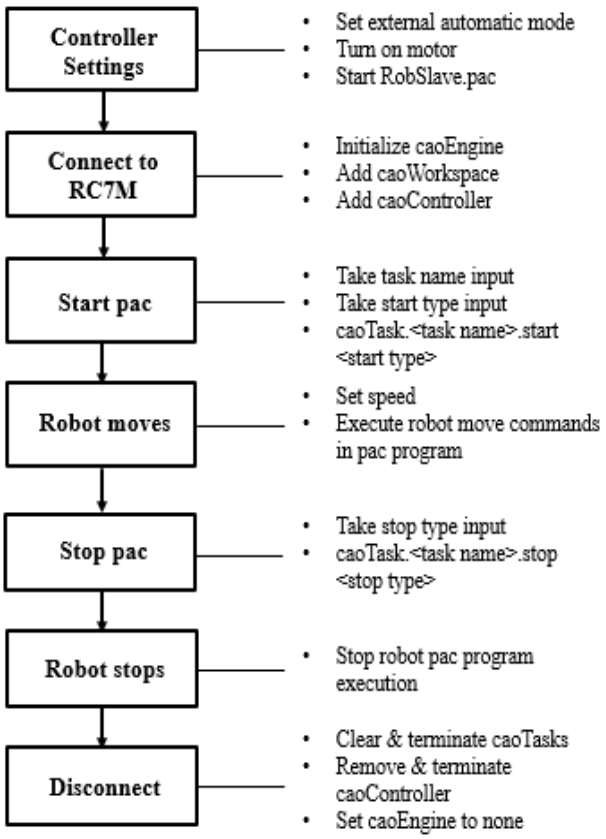

FIGURE 3. VB 6 test executable workflow.

IP address. Once the connection is established, the executable uses the input robotic arm program task name and start type as commands to run the prestored robotic arm program on the controller. This step serves the same purpose as the teach pendant we used before. The robotic arm program written in WINCAPS 3 can set the robot motion speed and execute through the test motions the robot needs to perform. After the robot motions complete or an interrupt (stop) command is received from the executable, the robot stops the motion, and the executable goes through the disconnect processes that terminates caoTasks and caoController, then removes these objects. The caoEngine is set to none, and the executable exits with the exit code.

D. MOTION-BASED IMAGE CAPTURE SOFTWARE OVERVIEW

The motion-based image capture software under test is called SmartSnap. The SmartSnap software can automatically capture the most active motion-based images by analyzing the information on each input frame. SmartSnap leverages the accelerated video encoding hardware engine generates metadata that contains a variety of image information, rather than regular video compression.

The algorithm flow for this software is shown in Figure 4. First, the camera obtains the input frames and passes them to the video codec engine to extract the metadata.

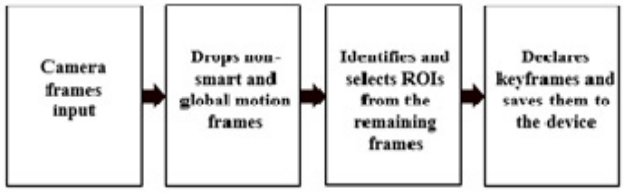

FIGURE 4. Motion-based image capture software algorithm overview.

The information identifies and separates the frames into Intra-frames and frames contain the motion related information. The intraframe elimination stage drops the Intra-frames as they are identified as ''non-smart snaps,'' Then, the global motion elimination stage drops the frames from the remaining frames such that all the vectors on the frame move, meaning that the motion is coming from the movement of the camera. In the next stage, the metadata goes through a series of computations and analysis to determine which regions from the frames are the regions of interest (ROIs) we are capturing, as well as how much the region of interest (ROIs) have changed in each frame. Then the algorithm determines if an ROI is a key ROI by determining whether all the metrics are above the thresholds. Finally, the frames that are determined to be key ROIs are declared as the keyframes (smart snaps), and they are saved to the device.

E. STRATEGY FOR AUTOMATED TESTING WITH SMARTSNAP

1) AUTOMATED TESTING FOR IMAGE CAPTURE LATENCY MEASUREMENT

Image capture latency is a key metric for motion-based image capture software. Whenever an interesting motion happens, regardless of the motion speed, the time duration of the motion can be short. Therefore, we do not want to have a long latency on an image capture trigger from a qualifying motion. In addition, the measurement of the latency on the motion that starts at the first image capture event becomes a critical KPI for this type of software testing.

The setup for automating this latency testing is simple. We have a display monitor or a projector that displays a video of some types of motion that will trigger the image capture, in this case, a video of a ball in downward motion. The device running the image capture software is held by the robotic arm. The robotic arm focuses the device with its field of view covering the displayed video. When the automation starts, the test script triggers the video to play, at the same time the script also triggers the start of the software's image capture process. This process should sense the ball's motion in the video and start the image capture at some time after the event as shown in Figure 5 below.

The test script also has the device's screen recorded during the testing process. After the test execution completes, it enters a postprocessing stage where the recorded video frames go through a MATLAB information extraction process. Through pattern matching, it determines

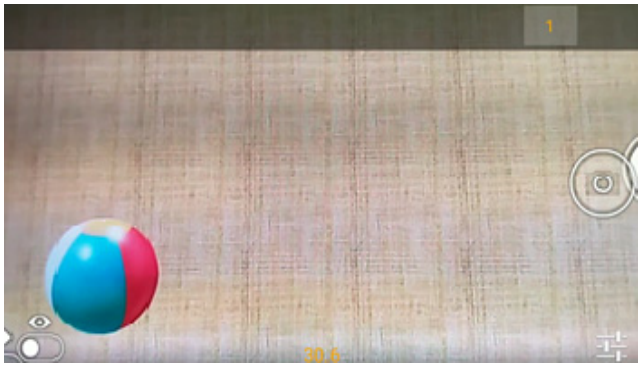

FIGURE 5. SmartSnap software view of the image capture process.

the frame number at which the ball starts moving, and the frame number of the first image captured indicated on the app. The total number of frames it captures divided by the frame rate of the video is a good approximation of the latency of the first image capture.

Using this testing method, the test coverage can be expanded by changing the motion speed of the moving object, as well as its size and appearance. This test will be useful when comparing the performance of different motion tracking software and validating the algorithm design by analyzing the latency changes with the test variations.

2) AUTOMATED TESTING FOR IMAGE CAPTURE ACCURACY MEASUREMENT

Image capture accuracy is another key metric for motion-based image capture software performance. If, under the condition that the test device remains stationary and the test subject performs the exact motion from time to time, then the software is supposed to capture the same number of image snapshots and these snapshots should be able to be paired based on the similarities from each run. From the number of images captured and their similarities, we can determine an overall accuracy score.

In our automated testing design, we chose to use 3D objects instead of 2D images to better simulate the real-life object motion. The motions were created by a programmable rotator that can be controlled by the test machine. It provides a precise turning motion with programmable speed and angles. In our test, we chose a female mannequin, a furry toy and a building model that represent different textured test subjects under motion. These test subjects were placed on top of the rotator and rotated to a precise angular degree from each test run as shown in Figure 6.

The test automation starts the image capture process on the software under test, then triggers the rotator turning motion. At the end of the test, the number of snapshots and all the snapshot images are collected and processed through postprocessing automation to obtain an accuracy percentage based on the resulting consistency from multiple test runs.

Automated testing for image capture accuracy with relative motions

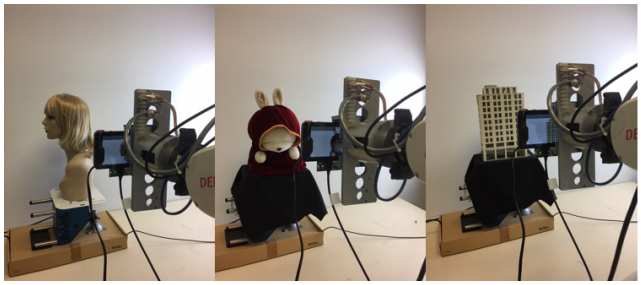

FIGURE 6. Test setup for image capture accuracy test automation.

In real-life motion-based image capturing scenarios, not only the objects move, but the people holding the camera and running the image capturing software also move. In this case, in the frames captured by the software, the background is also moving with the object at the same time. Therefore, we want to test the effect of device motion impacting the image capture accuracy.

The test is designed to have the subject remain stationary with a textured background behind the object. The device initially focuses on the subject where it stays in the center of the preview as shown in Figure 7.

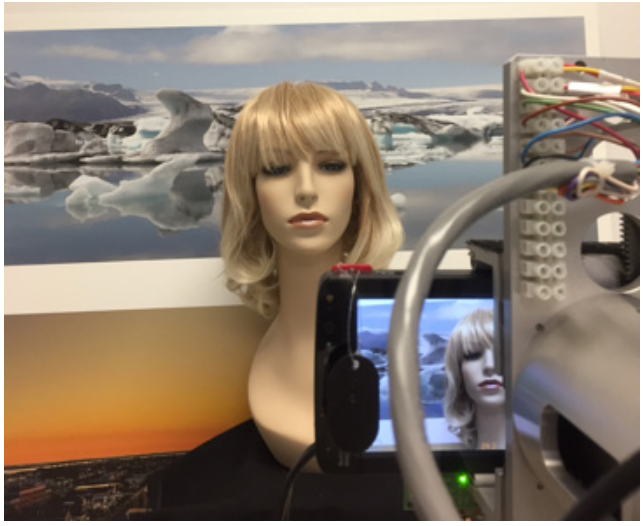

FIGURE 7. Test setup for image capture accuracy automation with relative device motions.

The device then moves relative to the test subject and the background so that it simulates a hand moving motion. The software should trigger an image snapshot based on the relative motion. This is where the robotic arm becomes useful. For the basic hand moving motions, we want the device to go up and down, left and right while the test subject stays in the field of view.

We can create a robotic arm program that performs these actions. In the WINCAPS 3 simulation mode, we create a 3D workspace of the robotic arm with a plane that is parallel to the yz plane. The distance between the plane and the origin should be the distance that the robotic arm extends out from the device. The distance is determined by the distance between the test subject and the test device. Then, we need to determine the y and z coordinates of the points at the four endpoints of the robotic arm motion. These four points should all land on the plane we simulated (in green) in Figure 8.

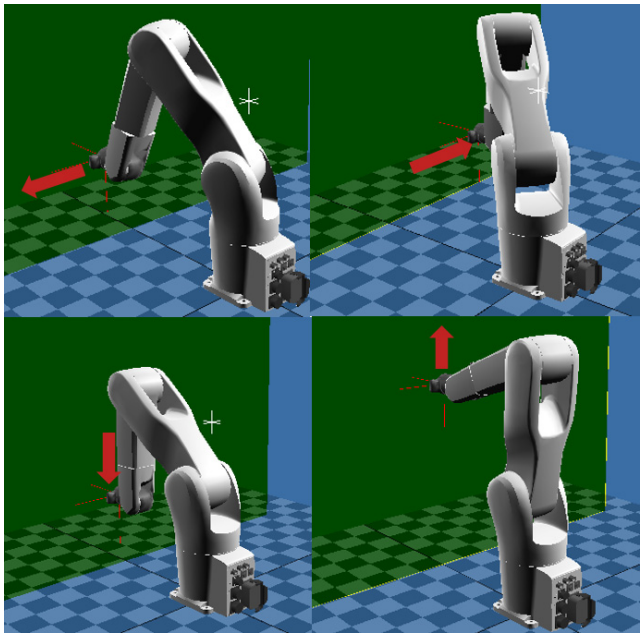

FIGURE 8. Simulation of the robotic arm motion in the workspace.

The four endpoints on the plane can be taught to the robotic arm so that the robotic arm calculates how to move to these coordinates. A robotic arm motion program can be created by commanding the arm to move between these coordinates to perform the motions we desire. The robotic arm's motion speed can also be specified inside or outside of the program. Therefore, the hand motion speed can be used as a test factor to study how it affects the image capture accuracy.

Similar to the automated testing for image capture accuracy, we use the same test subjects with the device motion instead. The test repeats for multiple runs with different speed settings. The results are postprocessed to obtain the accuracy results.

V. RESULTS

Using the integrated test automation system, the motion-based image capture testing is easily executed. Due to the advantages in the repeatability and the superior precision, we can use this system to perform competitive analysis on two similar motion-based image capture software performances and study their behaviors.

A. TEST RESULTS FOR IMAGE CAPTURE LATENCY MEASUREMENT

In this test, we used the SmartSnap image capture software and a 3rd-party motion-based image capture software for comparison. The automated image capture latency

measurement test ran one-hundred times on both software programs to obtain the averaged results. Figure 9 shows the segments of the image capture snapshots of a medium size ball with a downward motion and medium speed.

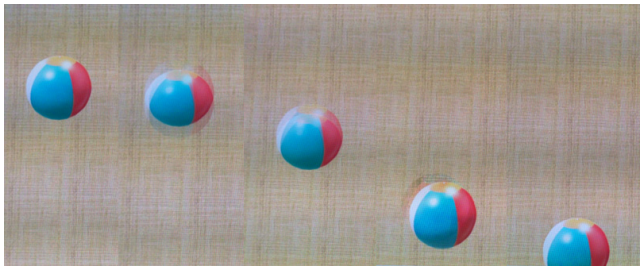

FIGURE 9. Segments of snapshots with a medium sized ball motion capture with medium speed.

From the snapshot images, we can see that the first ball image was captured with a slight delay after the ball started moving, as the position was not the initial position when the motion started. The measurements of these delays were found by processing the videos through a program written using the MATLAB programming language. The tests were repeated for small, medium and large sized balls moving at low, medium and high speeds. Table 1 shows the averaged latency results gathered through the automated testing for both software programs.

TABLE 1. The image capture latency measurement results from SmartSnap and the 3rd-party software.

Image Capture Latency Comparison Table			
Subject Size	Small Size	Medium Size	Large Size
Low Speed			
SmartSnap	0.42s	0.49s	0.52s
3rd-party	0.69s	0.75s	0.86s
Medium Speed			
SmartSnap	0.38s	0.42s	0.49s
3rd-party	0.41s	0.43s	0.58s
High Speed			
SmartSnap	0.36s	0.39s	0.43s
3rd-party	0.41s	0.44s	0.51s

From this results table, we can see that both of the software programs had averaged latency of less than one second on any of the test scenarios. In addition, in general, the SmartSnap software had better latency than the 3rd-party software. We plotted the data on a bar graph in Figure 10 to better visualize the differences.

The graph shows that the latency on the trigger of the image capture was at the highest at the low-motion speed, in general. As the motion speed became faster, the latency decreased. The 3rd-party software was affected considerably more by the low-speed motion factor. However, the

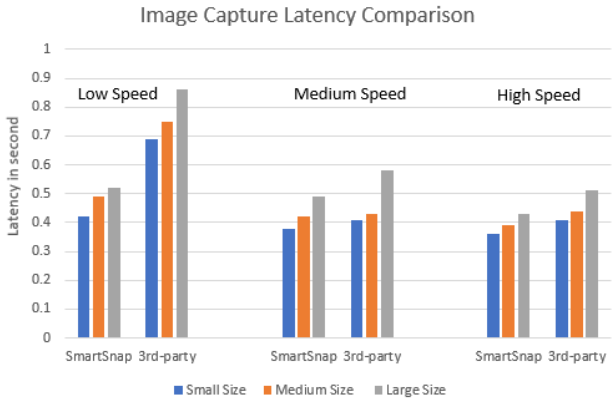

FIGURE 10. Graph of the image capture latency comparison.

SmartSnap software did not achieve a much better effect from the speed variation. Additionally, as the motion speed increased, the latency seemed to have a diminished return at high speed. This was likely due to the base processing time for the image capture being close to the measured time delay we found at high speed.

FIGURE 11. Text recognition result images from both applications under hand jitter scenarios.

B. TEST RESULTS FOR IMAGE CAPTURE ACCURACY MEASUREMENT

In the other test scenario, we measured the image capture accuracy from the SmartSnap and the 3rd-party software for one hundred iterations. We programmed the rotator to rotate at low, medium and high speeds for a 360-degree angle, using the mannequin, the furry toy and the building model as test subjects. As the test subjects rotated, which created motion, they triggered the software to start taking snapshots. Figure 11 shows the snapshots that were captured as the mannequin's head turned, capturing the different angles of the head.

After the automated tests were completed, all the results were averaged and collected in Table 2.

From the result table, we can see again that the SmartSnap showed better accuracy than the 3rd-party software. At slow speed, the accuracy was the lowest, and as the motion speed increased, the accuracy also increased. Figure 12 shows the bar graph using the data from this table.

The result graph shows a steady accuracy decrease across the test distribution from the mannequin to the furry toy and

TABLE 2. Image capture accuracy comparison result table.

Image Capture Accuracy Comparison Table			
Test Subjects	Mannequin	Furry Toy	Building Model
Slow Speed			
SmartSnap	75%	70%	66%
3rd-party	69%	63%	54%
Medium Speed			
SmartSnap	82%	81%	75%
3rd-party	75%	72%	63%
Fast Speed			
SmartSnap	91%	85%	80%
3rd-party	82%	77%	74%

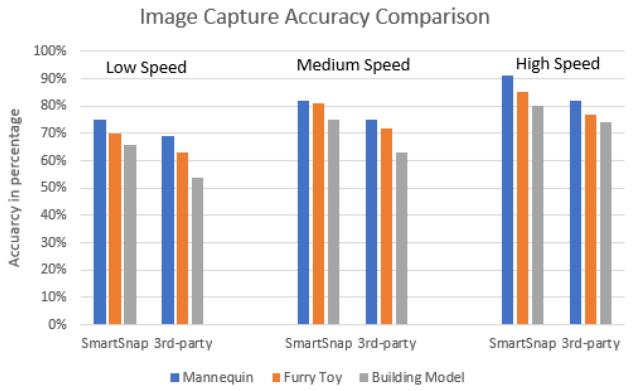

FIGURE 12. Bar graph of image capture accuracy comparison between the SmartSnap and the 3rd-party software.

TABLE 3. Image capture accuracy results table with relative motion.

Image Capture Accuracy with Relative Motion			
Test Subjects	Mannequin	Furry Toy	Building Model
Low Relative Motion Speed			
SmartSnap	72%	73%	70%
3rd-party	66%	65%	64%
Medium Relative Motion Speed			
SmartSnap	78%	80%	79%
3rd-party	75%	72%	73%
High Relative Motion Speed			
SmartSnap	88%	85%	85%
3rd-party	80%	78%	77%

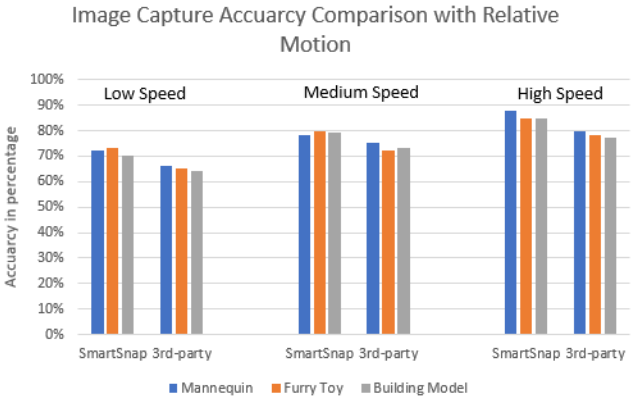

FIGURE 13. Bar graph of the image capture accuracy comparison under relative motion.

to the building model. The reason for this was likely due to the mannequin having a more unique texture distribution from the hair to the smooth skin region; the face portion also formed a different depth map which made it easier for the software to detect motion. On the furry toy, there was less feature distribution, and the whole body had a similar texture. This made it slightly more difficult for the software to detect motion. On the building model, almost all of the regions had the same texture. The building was also very symmetric and uniformly distributed, making it look the same from most angles. This could have caused the software to lose motion detection for some angles, resulting in snapshots being missed in the process.

C. TEST RESULTS FOR IMAGE CAPTURE LATENCY WITH RELATIVE MOTION MEASUREMENT

In the relative motion testing, we moved the test device to create motion instead of turning the test subjects. In addition, we followed the same test procedures from the previous scenario. The automation controlled the robotic arm to move the device at low, medium and high speeds. Images were captured during this process and triggered by the relative motion. The test results were collected in Table 3.

In this result table and the result graph from Figure 13, we found that the average results of the different test subjects were not creating the same level of delta in accuracy as in the previous test scenario. In some cases, different test subjects had very similar average accuracy results. This was due to the test subjects no longer being the only object that was moving and creating motion; instead, the test subjects were part of the background, and they were moving at the same time. This caused the test subjects to have little effect on triggering the motion. The only accuracy factor was the motion speed in this case. As expected, the higher the speed, the greater the accuracy. Additionally, in this case, the SmartSnap had slightly better accuracy than the 3rd-party software.

VI. CONCLUSION

The results of the motion-based image capture system based on a robotic arm show that the hardware-accelerated implementation performs better in terms of the results for the image capture latency with relative motion measurement, image capture accuracy measurement and image capture latency measurement compared to the 3rd party solutions. We observe an improvement of approximately twelve percent

in the image accuracy of the hardware accelerated solution compared with the third-party solution with medium panning speeds.

The deployment of the integrated solution of the test automation with the elimination of the robotic pendant for controlling the robotic arm movements and the usage of the ORiN2 middleware software to directly communicate from the PC to the robotic controller helped us to integrate the on-device test automation script with the executable executing the ORiN2 software. The integration helped us to programmatically sequence the on-device test execution and terminate the test automation in cases where we see fundamental failures in certain steps. For example, if the motion-based image capture application fails to launch on the device, we can terminate the test case and free the robotic arm resource from further execution. The detection of the failure to launch of the motion-based capture application can be performed after analyzing the logs of the application. This way, we can effectively save time and resources where there are fundamental failures and proceed with the test automation with much better resource utilization.

The deployment of the test automation has helped us to scale the test automation across multiple software product lines and ensure that we obtain reliable test execution results.

REFERENCES

[1] Z. Guo, W. Yang, M. Li, X. Yi, Z. Cai, and Y. Wang, "ALLIANCE-ROS: A software framework on ROS for fault-tolerant and cooperative mobile robots," *Chin. J. Electron.*, vol. 27, no. 3, pp. 467–475, May 2018.

[2] M. Quigley *et al.*, "ROS: An opensource robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, vol. 3, no. 3.2, 2009, p. 5.

[3] A. Belzunce, M. Li, and H. Handroos, "Control system design of a teleoperated omnidirectional mobile robot using ROS," in *Proc. IEEE 11th Conf. Ind. Electron. Appl. (ICIEA)*, Jun. 2016, pp. 1283–1287.

[4] L. Peppoloni, F. Brizzi, C. A. Avizzano, and E. Ruffaldi, "Immersive ROS-integrated framework for robot teleoperation," in *Proc. IEEE Symp. 3D User Interfaces (3DUI)*, Arles, France, Mar. 2015, pp. 177–178.

[5] A. Leeper, K. Hsiao, M. Ciocarlie, L. Takayama, and D. Gossow, "Strategies for human-in-the-loop robotic grasping," in *Proc. 7th ACM/IEEE Int. Conf. Hum.-Robot Interact. (HRI)*, Boston, MA, USA, Mar. 2012, pp. 1–8.

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

[7] K. Mao, M. Harman, and Y. Jia, "Sapienz: Multi-objective automated testing for Android applications," in *Proc. 25th Int. Symp. Softw. Test. Anal. (ISSTA)*, 2016, pp. 94–105.

[8] K. Mao, L. Capra, M. Harman, and Y. Jia, "A survey of the use of crowdsourcing in software engineering," *J. Syst. Softw.*, vol. 126, pp. 57–84, Apr. 2017.

[9] D. Banerjee and K. Yu, "Robotic arm-based face recognition software test automation," *IEEE Access*, vol. 6, pp. 37858–37868, 2018.

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

[11] P. H. L. Silva and A. M. F. Burlamaqui, "System for creation, programming and availability of distributed robot teams using collective knowledge," *IEEE Latin Amer. Trans.*, vol. 14, no. 10, pp. 4392–4401, Oct. 2016.

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

[13] D. Banerjee, K. Yu, and G. Aggarwal, "Object tracking test automation using a robotic arm," *IEEE Access*, vol. 6, pp. 56378–56394, 2018, doi: 10.1109/ACCESS.2018.2873284.

[14] V. Michna, P. Wagner, and J. Kotzian, "Methods of computer-assisted manual control of wheeled robots," in *Proc. Int. Multiconf. Comput. Sci. Inf. Technol.*, Wisla, Poland, Oct. 2010, pp. 813–816.

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

[16] Y. Wang, L. Gu, Y. Xu, and X. Cao, "Practical tracking control of robot manipulators with continuous fractional-order nonsingular terminal sliding mode," *IEEE Trans. Ind. Electron.*, vol. 63, no. 10, pp. 6194–6204, Oct. 2016.

DEBDEEP BANERJEE received the master's degree in electrical engineering from the Illinois Institute of Technology. He has over 10 years of industry experience in the field of software/systems engineering. He is currently a Senior Staff Engineer and an Engineering Manager at Qualcomm Technologies, Inc. He is also the Software/Systems Development Engineer Test Lead for the Computer Vision Project, where he is responsible for test automation design, planning, development, deployment, code reviews, and managing the project. He works closely with the software/system teams. He has been with the software testing automation team since the inception of the computer vision project at Qualcomm Technologies, Inc., where he is involved in managing and developing the robotic arm software used in the Computer Vision Lab.

KEVIN YU is currently a Test Engineer at Qualcomm Technologies, Inc., 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.